Département de géomatique appliquée

Faculté des lettres et sciences humaines

Université de Sherbrooke

Intégration d'un module d'apprentissage profond dans l'architecture logicielle d'un SIG Web

Paul Cornioley

Essai présenté pour l'obtention du grade de Maître en science (M. Sc.),

cheminement géodéveloppement durable

Mai 2018

Remerciements

Je tiens à remercier sincèrement mon superviseur d'essai, M. Mickaël Germain, pour m'avoir proposé ce projet passionnant. Je lui suis également très reconnaissant pour son engagement et sa disponibilité tout au long de ce travail. Son soutien et ses conseils avisés lors de nos discussions m'ont non seulement été d'une grande aide, mais ont également été une importante source d'inspiration.

Mes remerciements vont également à ma conjointe sans qui cette aventure n'aurait pas été possible et qui a su m'offrir son support inconditionnel et ses encouragements tout au long de mon retour aux études.

Table des matières

Table des figures

Table des tableaux

Liste des Annexes

Liste des abréviations

AJAX : *Asynchronous JavaScript and XML*

API : *Application Programming Interface*

CNN : *Convolutional neural network*

CNTK : *Microsoft Cognitive Toolkit*

CPU : *Central processing unit*

GPU : *Graphics processing unit*

IA : Intelligence artificielle

MIT : *Massachusetts Institute of Technology*

MLC : Maximum de vraisemblance

NAIP : *National Agriculture Imagery Program*

OGC : *Open Geospatial Consortium*

PIR : Proche infrarouge

ReLU : *Rectified Linear Activation*

RGB : Rouge, vert et bleu

SIG : Système information géographique

SVM : Machine à vecteurs de support

WFS : *Web Feature Service*

WMS : *Web Map Service*

1. Introduction

1.1 Contexte

Depuis plusieurs années, l'intelligence artificielle (IA) connaît une très forte croissance de popularité aussi bien dans le milieu de la recherche scientifique qu'auprès des grandes compagnies des technologies de l'information comme Google, Amazon, Microsoft ou Facebook. Cette technologie s'invite aujourd'hui dans de nombreux domaines tels que les moteurs de recherche sur le Web (Bloomberg, 2018), les assistants virtuels (Amazon.com Inc., 2018), la reconnaissance d'images (Taigman *et al.*, 2014) ou encore les voitures autonomes (Huval *et al.*, 2015). Cette popularité est due aux récents développements d'un domaine de l'intelligence artificielle : l'apprentissage profond (*deep learning* en anglais). Des avancées remarquables ont été faites dans la reconnaissance de parole (Hinton *et al.* 2012) et d'images (Krizhevsky *et al.*, 2012) en surpassant la précision des algorithmes utilisés jusqu'alors.

La majorité des algorithmes d'apprentissage profond sont basés sur des réseaux de neurones artificiels. Ceux-ci sont composés de neurones interconnectés organisés en couches successives qui contiennent des représentations des données à différents niveaux d'abstraction (LeCun *et al.*, 2015). Chaque couche contient un certain nombre de paramètres (des poids) qui sont ajustés lorsque le réseau est exposé à un grand nombre de données.

Depuis quelques dizaines d'années, les réseaux de neurones sont utilisés en géomatique, et plus particulièrement en télédétection. On peut notamment citer les travaux de Foody *et al.* (1995) qui ont utilisé un réseau de neurones artificiels pour classifier des cultures agricoles à l'aide d'images radar. Les recherches se multiplient ces dernières années et le potentiel de reconnaissance d'images est exploité, par exemple, à des fins de classification d'images à très haute résolution ou de reconnaissance de cibles (Zhang *et al.*, 2016).

De manière non apparentée, les progrès dans les technologies de l'Internet ont permis l'émergence d'un nouveau domaine de la géomatique : la cartographie sur le Web ou les systèmes d'information géographique (SIG) Web. La cartographie en ligne a su se démocratiser permettant à présent de créer des cartographies en utilisant des gabarits et fonctionnalités déjà développés. Si les SIG sur le Web n'ont pas apporté de nouvelles fonctionnalités en géomatique, ils ont en revanche amené une nouvelle approche au partage des données et une facilité d'interaction et de développement d'applications (Haklay *et al.*, 2008). La nécessité de rédiger du code a été grandement facilitée par

le développement d'API (*Application Programming Interface*). La création d'applications cartographiques devient ainsi à la portée d'une communauté beaucoup plus large. On peut notamment citer l'API cartographique de Google Maps qui, en 2005, a ouvert la voie à la cartographie en ligne (Peterson, 2012).

Ainsi, grâce à ces outils, il est possible de développer des applications de plus en plus sophistiquées qui se rapprochent au niveau des fonctionnalités d'applications SIG de bureau telles qu'ArcMap ou QGIS (Steiniger et Hunter, 2013). L'accès à la cartographie et à des outils SIG se démocratise ainsi puisqu'il n'est plus nécessaire d'installer un logiciel de bureau dédié pour accéder à certaines fonctionnalités SIG, et un simple navigateur Web suffit pour visualiser et interagir avec l'information géospatiale.

1.2 Problématique

Présentement, dans le milieu de la géomatique, l'apprentissage profond est principalement appliqué dans des projets de recherche. La communauté des géomaticiens n'a pas encore accès à l'apprentissage profond au même titre que d'autres outils d'analyse spatiale comme par exemple l'outil *GeoStatistical Analyst* d'ArcGIS (ESRI, 2018) qui permet notamment d'interpoler un semis de points avec les méthodes de krigeage.

Bien qu'il existe de nombreuses librairies d'apprentissage automatique qui facilitent l'utilisation de l'apprentissage profond, la mise en œuvre de réseaux de neurones peut être compliquée. Cela nécessite de bonnes connaissances en apprentissage profond afin de définir une architecture de modèle cohérente mais également des connaissances en programmation afin de manipuler les données et entrainer les modèles définis.

À ce jour, il existe très peu d'outils intégrés dans les solutions SIG les plus courantes qui permettent d'exploiter l'apprentissage profond. Seul le logiciel d'analyse d'images ENVI propose l'utilisation d'algorithmes de classification basés sur des réseaux de neurones (Harris, 2018a).

Les réseaux de neurones peuvent s'appliquer à différentes problématiques de télédétection et cet essai s'intéresse plus particulièrement à la classification d'images à très hautes résolutions spatiales.

1.3 Objectifs

L'objectif principal de ce travail est de développer un module d'apprentissage profond intégré à un SIG sur le Web. Basé sur une librairie de programmation d'apprentissage profond existante, ce module permettra la classification d'images en utilisant les réseaux de neurones à convolution. La conception de l'application devra être suffisamment générique pour pouvoir s'appliquer à d'autres thématiques et être ainsi accessible à la communauté pour des projets de géomatique qui pourraient bénéficier de l'apprentissage profond. Également, dans le but de rendre l'outil accessible, le module sera sous la forme d'une application Web afin de s'affranchir de tout logiciel de bureau spécifique puisqu'un navigateur Web sera suffisant pour utiliser l'application.

Afin d'atteindre l'objectif principal, les objectifs spécifiques suivants ont été formulés :

- intégrer un module d'apprentissage profond dans l'architecture logicielle d'un SIG Web composée d'un système de gestion de base de données à référence spatiale et d'une interface d'interaction et de visualisation;
- proposer un échange des données interopérable dans le respect des normes de l'OGC (Open Geospatial Consortium);
- mettre en place une solution ouverte et libre pour assurer la pérennité et l'évolution du système;
- mettre en œuvre une interface conviviale et accessible à la communauté des géomaticiens.

2. Cadre théorique

2.1 Le développement de l'intelligence artificielle et de l'apprentissage profond

Le concept d'intelligence artificielle a fait son apparition dans les années 1950 parallèlement aux développements des premiers ordinateurs et avec les travaux d'Alan Turing, qui est considéré comme un des pères de l'intelligence artificielle (Cooper et Van Leeuwen, 2013). Son article *Computing Machinery and Intelligence* (Turing, 1950) introduit le fameux test de Turing, dans lequel il pose la question de savoir si une machine est capable de « penser ».

Depuis, l'intelligence artificielle a connu de nombreux développements, et, dans les années 1970 – 1980, les premières formes de réseaux de neurones apparaissent sous la forme d'architectures multicouches pouvant être entrainées par une méthode de descente de gradient (Werbos, 1974, LeCun, 1985, Rumelhart *et al.*, 1986). Aujourd'hui, la plupart des algorithmes d'apprentissage profond reposent sur des réseaux de neurones.

Ainsi, le concept n'est pas nouveau mais c'est seulement depuis quelques années que le domaine de l'intelligence artificiel est revenu sur le devant de la scène pour trois raisons principales (LeCun *et al.*, 2015, Chollet, 2018). Tout d'abord, les algorithmes d'apprentissage profond demandent des ressources computationnelles très importantes puisque les réseaux sont souvent composés de très nombreuses couches et connexions. Les récents développements dans les processeurs graphiques (GPU) permettent à présent d'effectuer des traitements entre 5 à 10 fois plus rapidement qu'un processeur central d'ordinateur (CPU) (Chollet, 2018). Cela permet d'entrainer des modèles beaucoup plus importants. D'autre part, pour être performant, ces algorithmes nécessitent des bases de données considérables pour la phase d'apprentissage. C'est notamment grâce une base de données d'environ 1.2 millions d'images, qu'en 2012, pour la première fois, une méthode d'apprentissage profond a supplanté les méthodes traditionnelles lors de la compétition ImageNet de reconnaissance d'images. Le but de cette compétition était de classifier les 1.2 millions d'images en 1000 différentes catégories (Krizhevsky *et al.*, 2012). Aujourd'hui, avec la prolifération des données sur Internet, il existe des nombreuses sources de données permettant d'entrainer des réseaux de neurones. Par ailleurs, depuis la compétition ImageNet, les algorithmes d'apprentissage profond n'ont cessé d'être améliorés, ce qui amène à la troisièmement raison de la popularité de l'apprentissage profond, qui est les avancées en algorithmie. De nombreuses recherches s'intéressent à présent à ce domaine et les prédictions ne cessent d'être améliorées.

L'apprentissage profond est un domaine d'étude de l'apprentissage automatique (*machine learning*) qui est une branche de l'intelligence artificielle (figure 1).

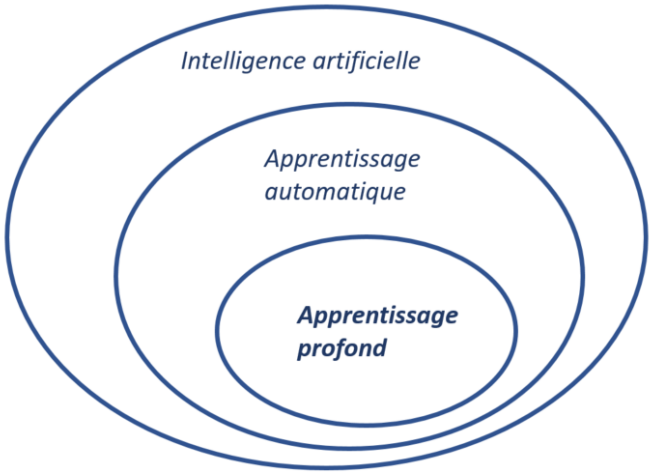


Figure 1 - Les domaines de l'intelligence artificielle, traduite de Chollet (2018).

Le principe de l'apprentissage automatique est que la machine est capable d'apprendre sur la base d'exemples afin de pouvoir par la suite répondre à des situations pour lesquelles elle n'a pas été explicitement programmée. Tout d'abord, la machine est entrainée : les données et leurs solutions sont fournies à la machine ce qui lui permet de déterminer des règles pour associer les données avec leur solution. Afin que la machine converge vers la meilleure solution, la performance est mesurée généralement à l'aide d'une fonction d'analyse. Ensuite, une fois la phase d'apprentissage terminée, la machine est confrontée à des données qu'elle n'a jamais vues et, grâce aux règles déterminées pendant la phase d'apprentissage, elle sera à même de faire des prédictions.

2.2 Fonctionnement des réseaux de neurones artificiels

La plupart des algorithmes d'apprentissage profond reposent sur des modèles appelés réseaux de neurones artificiels. Il existe de nombreuses variantes de ces réseaux qui ont été adaptées pour des problématiques spécifiques, et, par simplicité, les éléments communs seront présentés dans cette section. Par la suite, un type de modèle très utilisé pour la reconnaissance d'image sera présenté : les réseaux de neurones convolutifs.

Avant de présenter le fonctionnement d'un réseau de neurones, intéressons-nous à l'anatomie d'un réseau de neurones artificiels. Dans sa forme la plus simple, le plus petit élément d'un réseau est le neurone. Il prend la forme de plusieurs nœuds entrants qui sont connectés à un nœud de sortie (figure 2). Un poids est associé à chaque nœud entrant permettant de calculer une somme pondérée en multipliant chaque poids par la valeur des nœuds entrant. Le résultat de cette somme est transmis à une fonction d'activation qui va calculer la valeur de sortie de ce neurone. La valeur de sortie de ce neurone est ensuite transmise à un autre neurone.

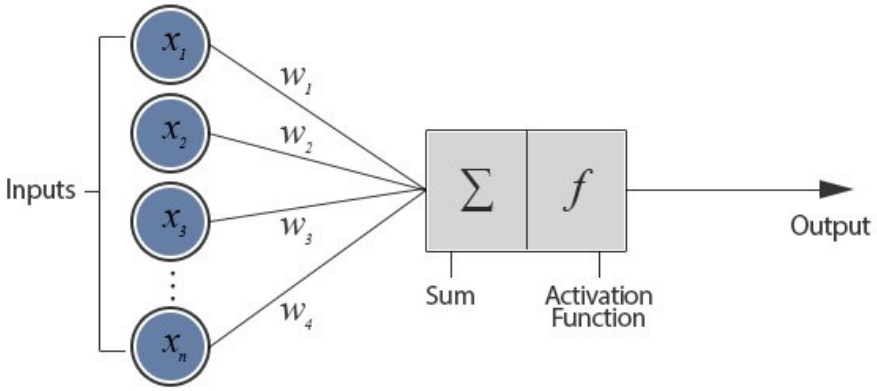


Figure 2 - Représentation d'un neurone artificiel, extraite et adaptée de DataCamp (2018).

La fonction d'activation est un élément essentiel aux réseaux de neurones. Elle introduit une non-linéarité dans le modèle et permet d'étendre la dimension de l'espace des hypothèses. Sans cette fonction d'activation, le modèle pourrait seulement apprendre des transformations linéaires ce qui restreindrait trop l'espace des hypothèses.

Les fonctions d'activation les plus courantes sont :

- La fonction ReLU (*Rectified Linear Activation*)
- La fonction sigmoïde

La fonction ReLU (figure 3) permet d'éviter les valeurs négatives à la sortie du neurone puisqu'elle met à zéro toutes les valeurs négatives alors que les valeurs positives sont inchangées. C'est une des fonctions d'activation les plus utilisées dans les réseaux de neurones. La fonction sigmoïde (figure 3) retourne une valeur comprise entre 0 et 1. C'est une fonction d'activation souvent utilisée pour la dernière couche afin d'obtenir à la sortie du réseau un score qui peut s'interpréter comme une probabilité.

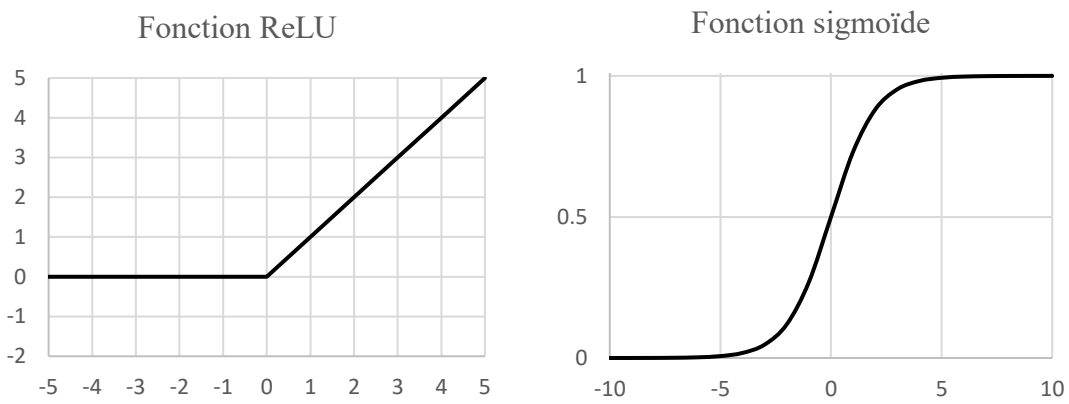


Figure 3 - Fonctions d'activation les plus courantes dans les réseaux de neurones artificiels.

Un réseau de neurones artificiels est une chaine de neurones organisée en couches (figure 4). La profondeur d'un réseau fait référence au nombre de couches qui est souvent très important. Le réseau est composé d'une couche en entrée et une couche en sortie qui sont dites, visibles. Entre ces couches de neurones, les couches sont dites cachées. Les neurones sont le plus souvent interconnectés, ce qui signifie que les neurones de chaque couche sont connectés à tous les neurones de la couche suivante.

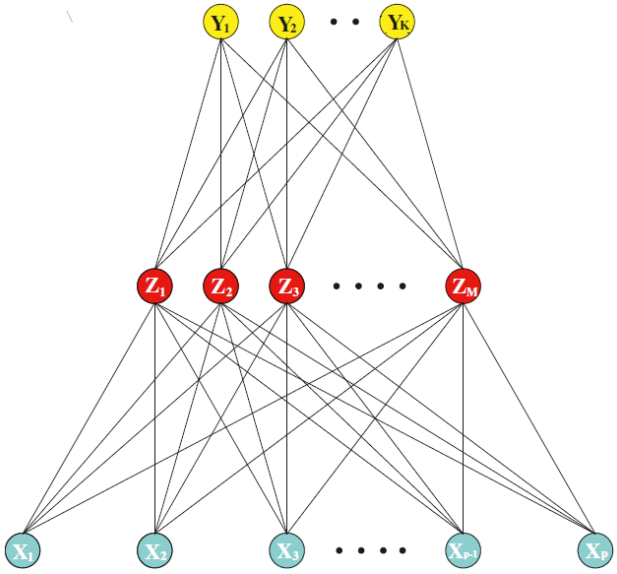


Figure 4 - Schéma d'un réseau de neurones interconnectés avec une seule couche caché, extraite de Hastie et al. (2009).

Comme pour l'apprentissage automatique classique, les réseaux de neurones nécessitent une phase d'apprentissage. Celle-ci est représentée à la figure 5, qui montre les interactions entre les différents composants d'un réseau de neurones lors de cette phase.

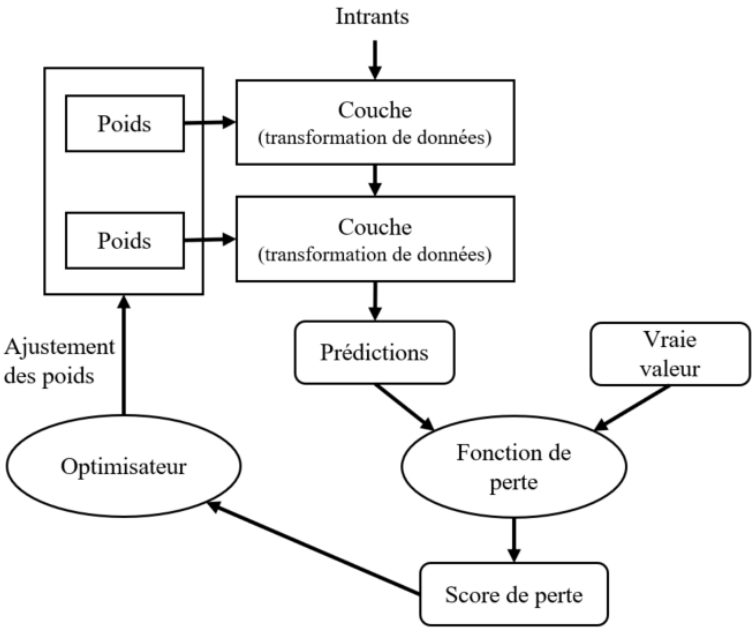


Figure 5 - Principaux composants d'un réseau de neurones artificiels, traduite de Chollet (2018).

Pendant la phase d'apprentissage (ou d'entrainement), le réseau de neurones est exposé à des données (*Intrants*) dont la solution est connue. À chaque couche, les données sont transformées et une représentation des données est créée. La dernière couche capture les interactions les plus complexes.

Une fois que la donnée est passée à travers toutes les couches, une prédiction est faite pour cette donnée. Cela peut être, par exemple, une seule valeur dans le cas d'un problème de régression, ou encore un score d'appartenance à chaque classe dans le cas d'un problème de classification à plusieurs classes.

La prédiction est ensuite confrontée à la vraie valeur de la donnée en utilisant une fonction de perte. Ceci permet d'obtenir un score de perte qui sert à évaluer l'écart d'erreur entre la valeur prédite et la vraie valeur. Ce score est utilisé par l'optimisateur qui permet d'ajuster les poids dans la bonne direction pour diminuer l'écart entre la prédiction et la vraie valeur. Ainsi, les poids du réseau servent à stocker l'apprentissage du modèle. Ce processus est appelé *backpropagation*.

Ensuite, les données sont à nouveau exposées au réseau de neurones et les étapes décrites ci-dessus sont répétées un nombre de fois défini. Si la précision obtenue après l'entrainement n'est pas jugée suffisante, cela peut signifier que l'architecture du modèle doit être modifiée. Il est souvent nécessaire de répéter la phase d'apprentissage plusieurs fois avant de trouver une architecture qui modélise correctement le jeu de données. Cela consiste notamment à ajouter ou enlever des couches et modifier leurs paramètres.

La fonction de perte et l'optimisateur sont deux composantes importantes du réseau et permettent de paramétriser la phase d'apprentissage. La fonction de perte sert à mesurer la performance de l'algorithme et produit le score de perte qu'il faut minimiser. Cette fonction confronte les valeurs de prédictions aux vraies valeurs des données. Le choix de cette fonction est une décision importante dans un modèle d'apprentissage profond puisqu'elle va mesurer le succès de la classification. Son choix dépendra du problème à l'étude (classification binaire, multi-classes, régression, etc.). Les fonctions les plus courantes selon les types de problèmes sont montrées dans le tableau 1.

Tableau 1 - Fonctions de perte les plus courantes selon les types de problèmes, traduit et adapté de Chollet (2018).

Types de problèmes	**Fonctions de perte**
Classification binaire	binary_crossentropy
Multi-classes, une seule étiquette possible	categorical_crossentropy
Multi-classes, plusieurs étiquettes possibles	binary_crossentropy
Régression vers des valeurs arbitraires	mse (*mean squared error*)
Régression vers des valeurs comprises entre 0 et 1	mse ou binary_crossentropy

Le choix de l'optimisateur influence également beaucoup la performance du modèle, c'est lui qui va permettre de mettre à jour les poids dans une direction qui va minimiser la fonction de perte. Il est basé sur un algorithme du gradient stochastique. C'est une méthode fondée sur les dérivées qui permet de minimiser une fonction objective.

Pour entrainer un modèle d'apprentissage profond, il faut un jeu de données dont les solutions sont connues. Par exemple, dans le cas d'une classification d'images, le jeu de données est composé des images et de leurs classes. Généralement, les données sont divisées en trois groupes :

- **Données d'entrainement** : données utilisées pour entrainer le modèle et ajuster les poids.
- **Données de validation** : données utilisées pour valider le modèle lors de l'entrainement pour vérifier qu'il n'y a pas de surapprentissage. Ces données ne sont pas utilisées pour ajuster les poids, seulement pour évaluer le modèle pendant l'apprentissage.
- **Données de test** : données utilisées une fois que le modèle a terminé d'être entrainé. Ceci permet de mesurer avec le plus de neutralité possible la précision du modèle. Lors de la phase d'apprentissage, l'architecture du modèle est adaptée et modifiée par l'utilisateur pour minimiser la perte des données de validation et cela peut mener à un surapprentissage pour ces données. Ainsi, afin de s'assurer de ne pas biaiser l'évaluation de la précision du modèle, elle est évaluée sur des données que le modèle n'a jamais vues.

Souvent, deux problèmes peuvent apparaitre lors de l'entrainement d'un modèle :

- **Surapprentissage** (*overfitting*) : le modèle est suroptimisé pour les données d'entrainement mais il performe piètrement avec de nouvelles données. Cette situation est présentée à la figure 6, la perte diminue pour les données d'entrainement avec le nombre d'époques alors que la perte des données de validation diminue puis remonte nettement. Cela signifie que le modèle se spécialise dans les données d'entrainement et qu'il n'est pas suffisamment générique avec d'autres données. Cela peut arriver lorsque le modèle possède trop de paramètres et de couches et donc la dimension de l'espace des hypothèses est trop grande. Dans le cas extrême, le modèle va apprendre à faire le lien entre chaque donnée d'entrainement et sa solution. Ainsi, il sera incapable de faire de bonnes prédictions avec de nouvelles données et le but de créer un modèle est justement de pouvoir faire des prédictions. C'est donc un problème à éviter absolument. Il peut être empêché en arrêtant simplement l'entrainement du modèle après un certain nombre d'époques, par exemple, à la figure 6, en arrêtant l'entrainement après environ 10 époques. Une meilleure solution est d'utiliser une succession spécifique de certains types de couches qui empêchent le surapprentissage.

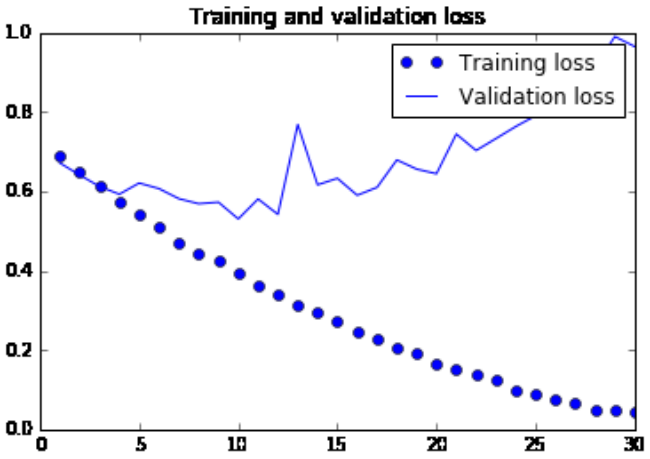


Figure 6 - Exemple de surapprentissage. Training loss : valeur de perte pour les données d'entrainement. Validation loss : valeur de perte pour les données de validation.

- **Minimum local** : lors de l'optimisation, il peut arriver que la fonction d'optimisation tombe dans un minimum local (figure 7). Cette situation est généralement évitée avec les fonctions d'optimisation les plus modernes.

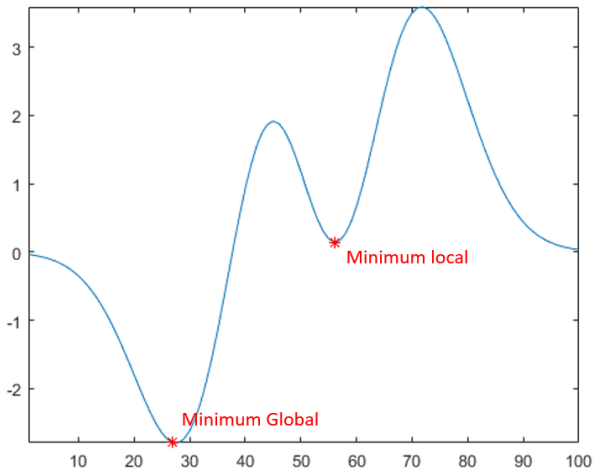


Figure 7 - Minimum local et global d'une fonction, adaptée de The MathWork Inc. (2018).

2.2.1 Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs (CNN) sont un type de réseaux de neurones particulièrement adaptés pour le traitement d'images car ils prennent en entrée des données matricielles à multiples dimensions comme les images (LeCun *et al.*, 2015). En effet, une image RGB est composée de trois matrices (une par bande spectrale) de deux dimensions (hauteur et largeur).

Le succès des CNN tient à deux caractéristiques centrales (Chollet, 2018). D'une part, ils sont capables d'apprendre des motifs locaux invariants comme des contours, des angles, de la texture, etc. Ainsi, si le motif est appris à un endroit de l'image, il sera reconnu à n'importe quel autre endroit de l'image, le modèle n'a pas besoin de l'apprendre à nouveau. Et, d'autre part, ils sont capables d'apprendre des hiérarchies spatiales de motifs : les premières couches du réseau vont apprendre des motifs simples et petits, et, plus les couches sont avancées, plus les motifs seront complexes et grands et formés des motifs simples appris précédemment.

En pratique, les CNN reposent sur deux types de couches successives, qui sont représentés à la figure 8 (LeCun *et al.*, 2015) :

- Couche de **convolution** : la couche permet d'identifier des motifs locaux à différents endroits de l'image.
- Couche de ***pooling*** : la couche combine les caractéristiques similaires ensembles.

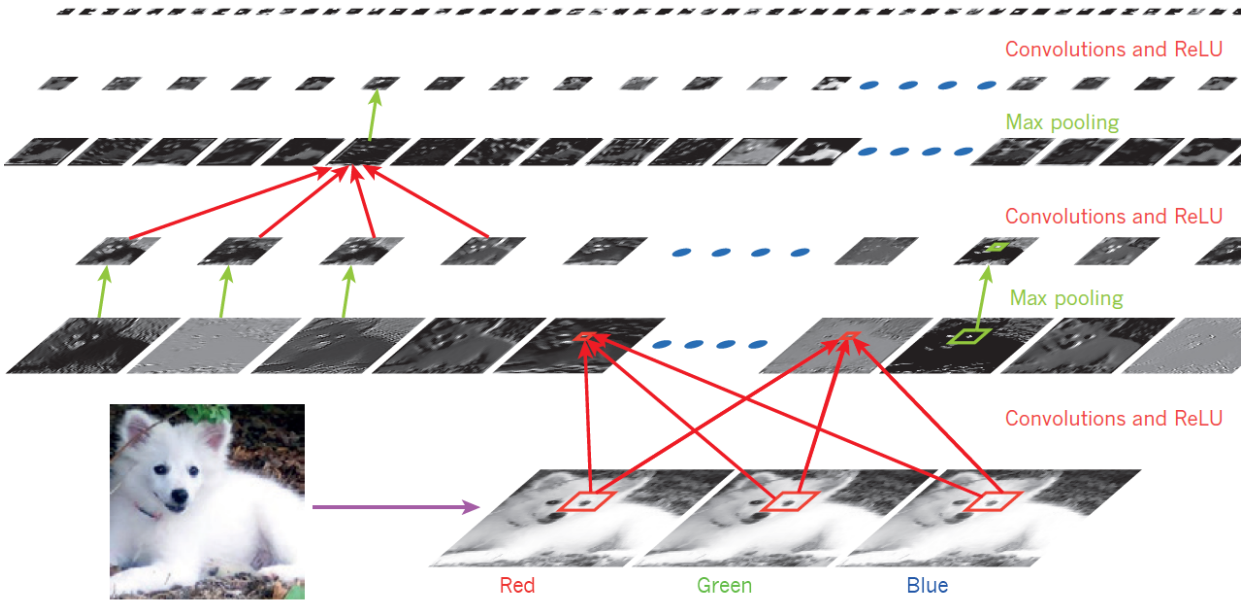


Figure 8 - Images de sortie des couches d'un réseau de neurones convolutifs montrant les étapes de convolution et de pooling, extraite de LeCun et al. (2015).

Pour entrainer un CNN, il est nécessaire de formater les images dans des matrices multidimensionnelles, aussi appelées tenseurs. Généralement, les images d'un jeu de données sont regroupées dans un tenseur de dimension 4 (figure 9) : le premier axe contient les entités, puis la hauteur de l'image, la largeur et le nombre de bandes. À noter que selon la convention, l'axe du nombre de bandes se trouve en deuxième position : entité, bande, hauteur, largeur.

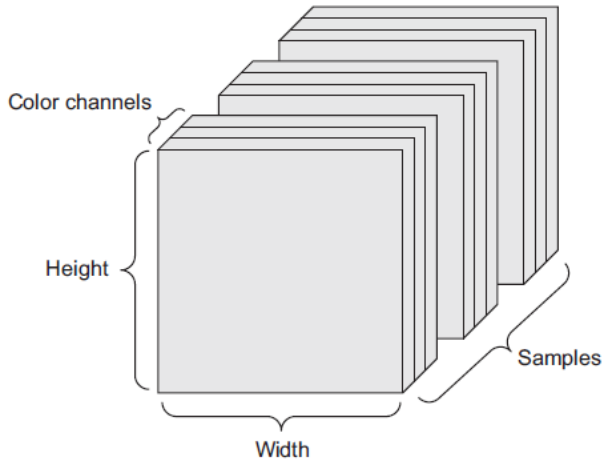


Figure 9 - Images sous forme de tenseur 4D avec la convention : entité, bande, hauteur, largeur. Extraite de Chollet (2018).

2.3 L'apprentissage profond et la géomatique

Les progrès de la vision par ordinateur dans les réseaux de neurones ont ouvert de nouvelles perspectives pour la géomatique et plus précisément pour la télédétection. Les recherches dans ce domaine se multiplient avec des résultats très encourageants. En effet, les CNN étant très performants avec des données images, ils peuvent être appliqués aussi bien à des images satellites qu'aéroportées. Par exemple, pour la reconnaissance de cibles, Tang *et al.* (2015) ont utilisé un réseau de neurones pour identifier des bateaux sur des images satellites. Les images ont d'abord été segmentées afin d'isoler les bateaux potentiels puis un réseau de neurones a été entrainés pour reconnaître les caractéristiques des bateaux.

Les réseaux de neurones ont aussi montré de très bons résultats pour la classification de l'occupation du sol. Par exemple, Basu *et al.* (2015) ont constitué une base de données composées de tuiles d'images aéroportées. À l'aide de celles-ci, ils ont obtenu des scores de classification avec un taux de succès jusqu'à 97,95%.

Dans une autre application intéressante de l'apprentissage profond, Bouroubi *et al.* (2017) ont utilisé les réseaux de neurones pour détecter les toits en vue de déterminer le potentiel photovoltaïque de Hô-Chi-Minh-Ville au Vietnam. Sur la base d'images de WordView-3, la photogrammétrie a notamment été utilisée afin d'extraire un modèle numérique de surface pour aider dans la détection des toits.

Par ailleurs, au sein du département de géomatique appliqué de l'université de Sherbrooke, des projets de recherche utilisant l'apprentissage profond sont en cours depuis l'année 2017. Notamment, deux projets de doctorats, sous la supervision du professeur Jérôme Théau, s'intéressent à l'analyse d'images à très haute résolution provenant de drones. L'un de ces doctorats se penche sur les inventaires fauniques et souhaite utiliser l'apprentissage profond afin de détecter la présence de cervidés de manière automatique en utilisant des réseaux de neurones convolutifs. L'autre thèse en cours s'intéresse à l'agriculture de précision et cherche à identifier les problèmes phytosanitaires de la vigne à partir d'images à très haute résolution.

2.4 Librairies d'apprentissage profond les plus courantes

Devant l'intérêt grandissant pour l'apprentissage profond, de nombreuses solutions libres et ouvertes ont vu le jour. L'éventail des solutions de programmation est assez large et évolutif. Les grands groupes des technologies de l'information investissent beaucoup de ressources dans ces solutions. Les enjeux tournent beaucoup autour des performances proposées, la facilité d'utilisation et la flexibilité offerte dans mise en place des modèles. Le langage Python est le langage de programmation le plus populaire dans l'utilisation de l'apprentissage profond. Le tableau 2 présente les librairies les plus courantes.

Tableau 2 - Librairies d'apprentissage profond les plus courantes.

Nom	Type de licence	Langages supportés	Développée par	Commentaires
Caffe	Libre et ouverte	Python, C++, Matlab	Berkeley AI Research (BAIR)	Développée par BVLC (Berkeley Vision and Learning Center), c'est une librairie très connue et performante pour le traitement d'images.
Theano	Libre et ouvert	Python	Algorithmes d'Apprentissage de Montréal de l'université de Montréal (MILA)	Librairie historiquement très connue et mais qui n'est plus développée.
TensorFlow	Libre et ouverte	C++, Python	Google Brain project	Similaire à Theano et la remplace d'une certaine façon, c'est une librairie de bas niveau.
Keras	Licence MIT	Python	François Chollet	Librairie minimaliste qui a pour but de simplifier la tâche du développeur en permettant l'expérimentation le plus rapidement possible.
PyTorch	Libre et ouverte	Python	Facebook AI Research (FAIR).	Librairie performante qui se distingue de TensorFlow dans sa façon de créer des graphes dynamiquement.
Matlab Neural network	Propriétaire	Matlab	The MathWorks Inc.	Solution propriétaire de Matlab.

Présente depuis plus de 4 ans (Jia *et al.*, 2014), **Caffe** est une librairie développée par le laboratoire de recherche en intelligence artificielle de Berkeley (BAIR, Berkeley Artificial Intelligence Research) (BAIR, 2018). Elle est utilisée dans de nombreuses recherches en intelligence artificielle et est devenue une librairie incontournable dans le domaine. Elle est particulièrement réputée pour ses hautes performances dans le traitement d'images. Elle est capable de traiter 60 millions d'images par jour avec un seul GPU Nvidia K40 (BAIR, 2018). Bien que cette librairie soit développée en C++, il est possible de l'utiliser dans le langage Python ou avec Matlab. Malgré ses bonnes performances et sa réputation, sa prise en main demande une courbe d'apprentissage assez importante et la manipulation des modèles n'est pas très conviviale (PyImageSearch, 2018). À noter qu'une version 2 (Caffe2) vient de paraître et qui est plus légère.

Theano (Lisa Lab., 2018) est une autre librairie incontournable. Apparue en 2007 et développée par l'Institut des Algorithmes d'Apprentissage de Montréal de l'université de Montréal (MILA), elle est une des premières librairie Python dans le domaine et elle a encouragé le développement de nombreuses autres libraires. Theano est une librairie de bas niveau, c'est-à-dire, qu'elle se concentre sur la manipulation et les traitements mathématiques des tenseurs qui sont utilisés dans les réseaux de neurones. En particulier, elle optimise ces opérations intensives en utilisant soit la puissance de calcul du GPU ou du CPU. Bien qu'il soit possible de définir des modèles à l'aide de cette librairie, cela peut être fastidieux. Theano est plutôt utilisée comme moteur d'arrière-plan (*backend engine*) par d'autres libraires. À noter que le MILA a cependant annoncé qu'à partir de la version 1.0.0 de Theano, il ne développera plus cette librairie (Lambin, 2017). Étant une librairie libre et ouverte, elle continuera d'être disponible mais il n'y aura plus de nouvelles implémentations de la part de MILA. Ceci démontre la forte évolutivité du domaine et d'une compétition très présente notamment avec d'autres acteurs de l'industrie.

Initialement apparue en 2015, la librairie **TensorFlow** est un autre acteur majeur (Google Brain, 2018). TensorFlow est une librairie libre et ouverte développée par le projet de recherche Google Brain. Similaire à Theano (qu'elle remplace d'une certaine manière depuis l'annonce de la fin de celle-ci), c'est une librairie de bas niveau et qui se concentre sur la manipulation et les opérations au niveau des tenseurs dans les réseaux de neurones.

Également liée à Google, **Keras** est une librairie libre et ouverte (licence MIT) développée en Python par un ingénieur de Google, François Chollet, à qui l'on doit également un ouvrage

accessible pour débuter dans l'apprentissage profond : *Deep Learning with Python* (Chollet, 2018). Keras a été développée dans le but de faciliter l'expérimentation. C'est une librairie de haut niveau particulièrement adaptée pour développer ses propres modèles grâce à un codage simplifié. Il est possible d'utiliser plusieurs moteurs en arrière-plan, dont TensorFlow (par défaut) et Theano, sans modifier le code. C'est donc une librairie très accessible notamment pour débuter en apprentissage profond.

PyTorch (PyTorch, 2018) est une autre librairie très populaire, maintenue par le groupe de recherche en intelligence artificielle de Facebook, elle a été publiée de manière libre et ouverte en 2017 (Skymind, 2018). Elle se distingue notamment d'autres librairies comme TensorFlow par sa manière de définir les modèles d'apprentissage qui est dynamique. Le graphe est créé en même temps que le modèle est entrainé, alors qu'avec TensorFlow, le graphe est d'abord créé puis entrainé.

Finalement, on peut également mentionner Matlab qui est un logiciel de programmation très présent dans le domaine de l'ingénierie et des mathématiques. Néanmoins, le logiciel est propriétaire de la compagnie Mathworks. Il existe aussi d'autres libraires libres et ouvertes comme Lasagne, MXNet, nolearn, etc. (PyImageSearch, 2018) mais il est impossible de les couvrir toutes dans le cadre de cet essai.

2.5 Classification d'images

La classification d'images est un problème récurrent en télédétection dont le but est d'identifier la classe à laquelle appartient chaque pixel d'une image. La classe représente un type d'occupation du sol (ou couverture du sol), par exemple : agriculture, urbain, eau, forêt, etc.

Il y a deux principales formes de classification :

- la classification non dirigée;
- et la classification dirigée.

Avec l'approche non dirigée, l'algorithme de classification va trouver des séparations entre les classes en se basant sur les caractéristiques statistiques et/ou spectrales des pixels. L'utilisateur peut imposer le nombre de classes à identifier et de raffiner ce nombre si nécessaire selon le résultat de la classification (Belspo, 2018).

La classification dirigée suppose que les classes recherchées dans l'image et leurs propriétés statistiques et/ou spectrales sont connues. Généralement, cette connaissance est obtenue par une campagne sur le terrain. Des zones représentatives des classes sont visitées ce qui permet de savoir avec certitude à quelles classes ces zones appartiennent, qui sont appelées des sites d'entrainement.

Il existe de nombreuses méthodes et algorithmes de classification, qu'on peut diviser en deux catégories principales :

- Classification par pixel : utilise l'information caractéristique des pixels.
- Texture : utilise l'information du voisinage des pixels.

La classification d'images par pixel repose sur l'analyse de la signature statistique et/ou spectrale des objets au sol. Dans les algorithmes de classification les plus courants, on peut citer :

- Maximum de vraisemblance (MLC)
- Distance Minimale
- Machine à vecteurs de support (SVM)
- Arbres aléatoires

Les logiciels SIG les plus courants comme ArcMap (ESRI, 2015) et QGIS (Free Software Foundation, Inc., 2016) permettent de classifier des images avec certains de ces algorithmes. Il y a aussi des logiciels spécialisés dans l'analyse d'images tels que PCI Geomatica (PCI Geomatics, 2018) ou ENVI (Harris, 2018b).

Dans les méthodes utilisant l'analyse de la signature spectrale, peu d'importance voire aucune n'est donnée aux pixels voisins. Néanmoins, pour tenir compte du contexte des pixels, il est possible de combiner la classification par pixel par une segmentation de l'image au préalable. La segmentation est une façon de diviser l'image par objet. Elle est basée sur une analyse statistique du voisinage des pixels.

La classification d'images par pixel produit de très bons résultats pour des images satellitaires à moyenne résolution spatiale (plus de 1 mètre) mais pour les images à très haute résolution spatiale (moins de 1 mètre) ces algorithmes commencent à montrer certaines limites, la variabilité du signal à l'intérieur des classes devient très importante ce qui rend la classification plus difficile (Carleer *et al.*, 2005).

Ainsi, pour des images à très haute résolution spatiale, la classification par pixel n'est plus adaptée et il faut tenir d'avantage compte du voisinage des pixels. Plusieurs développements ont eu lieu dans la classification utilisant la texture, on peut par exemple citer les matrices de cooccurrence, l'analyse fractale ou encore le spectre de texture.

3. Matériel et méthodes

3.1 Site d'étude

Le site d'étude se trouve dans deux zones géographiques distinctes : une ville du Québec (Sherbrooke) et un petit village en Suisse (Merlischachen). Ces régions sont décrites dans les sous-sections ci-dessous.

3.1.1 Sherbrooke

Sherbrooke est une ville du Canada située dans la province de Québec. Sa superficie est de 366,16 km^2 et sa population de 165 859 (Ville de Sherbrooke, 2018). Sa densité est ainsi de 453 habitants/km^2, c'est une ville peu dense située dans un milieu à tendance rurale, en comparaison, la ville de Montréal possède une densité de population de 3 891 habitants/km^2 (Ville de Montréal, 2016).

La zone d'étude qui nous intéresse plus particulièrement est celle délimitée par l'image de WorldView-3 (figure 10). Celle-ci englobe une partie des arrondissements de Mont-Bellevue, Rock Forest-Saint-Élie-Deauville et Jacques-Cartier. Une grande partie du site est couverte par des forêts et quelques champs agricoles. Deux zones résidentielles sont distinctes, au nord-est, dans l'arrondissement de Mont-Bellevue et à l'ouest, dans l'arrondissement de Rock Forest-Saint-Élie-Deauville.

Le site d'étude est traversé par la rivière Magog qui s'écoule du sud vers le nord et qui est séparée aux deux tiers de son parcours dans la zone d'étude par le barrage hydroélectrique Drummond. Le site est également traversé par plusieurs routes importantes dont l'autoroute Jacques-O'Bready et la route 216. On note également la présence de la carrière Bel Horizon dans la partie sud-est du site.

3.1.2 Merlischachen

Merlischachen est un village situé en Suisse dans le canton Schwytz, dans la Suisse centrale (figure 11). Il fait partie de la municipalité Küssnacht qui regroupe plusieurs villages totalisant 12 512 habitants (FSO, 2018).

La superficie de la zone à l'étude est de 0,15 km^2. Le site contient une petite zone résidentielle, une zone couverte d'herbe dans sa partie est, quelques champs au sud et quelques arbres dans la partie nord-ouest. Le site est traversé par une route principale du sud-ouest vers le nord-est.

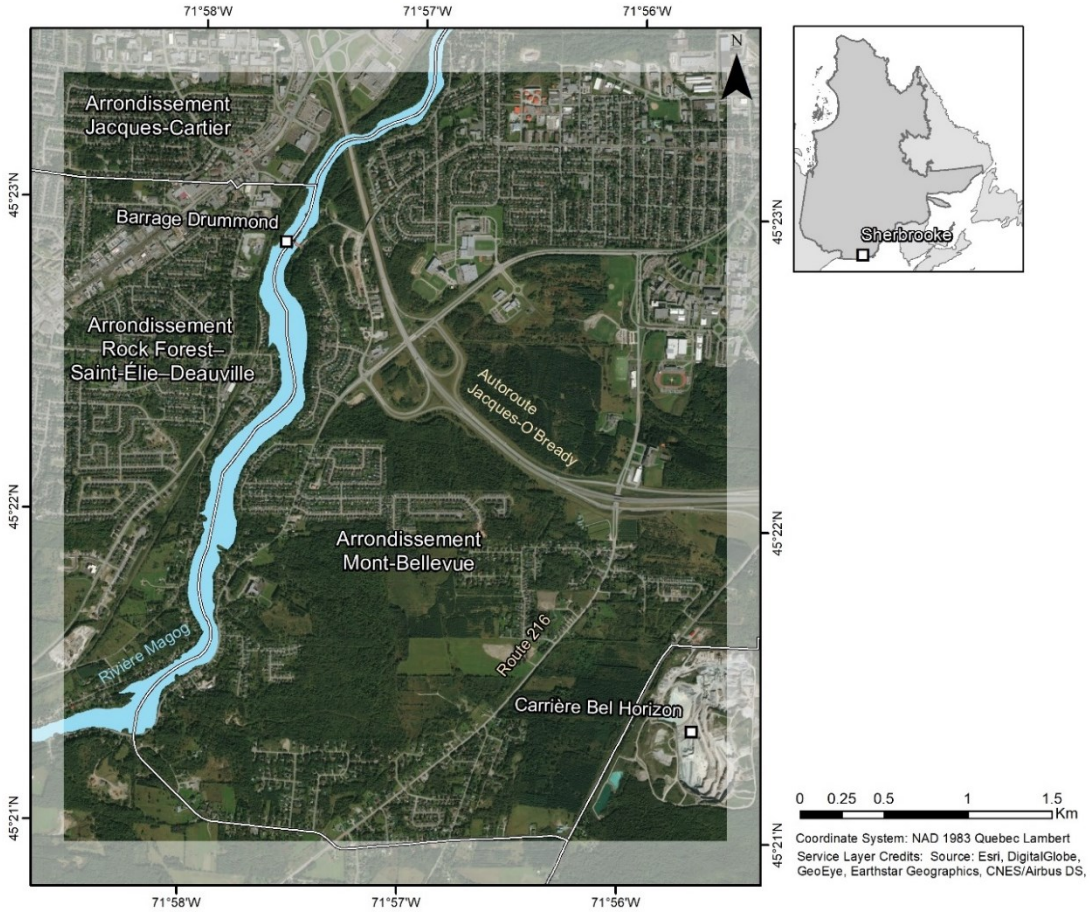


Figure 10 - Site d'étude de la ville de Sherbrooke.

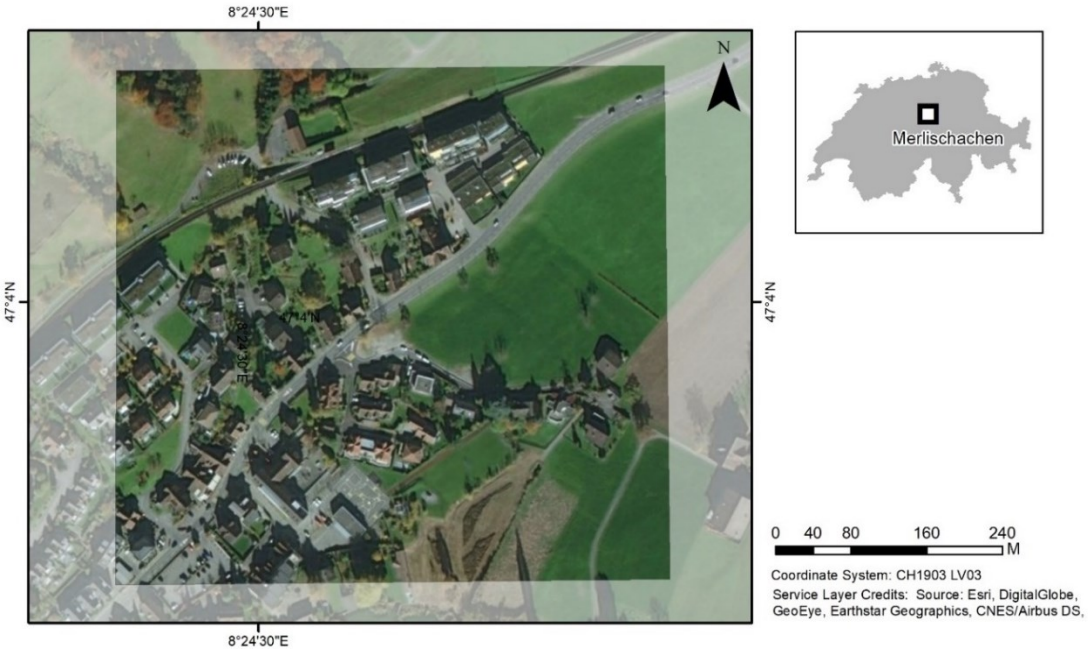


Figure 11 - Site d'étude du village Merlischachen.

3.2 Données

3.2.1 Données d'entrainement

Les données d'entrainement proviennent de la plateforme Kaggle (Kaggle, 2018). C'est une ressource bien connue de la communauté d'analystes de données, particulièrement dans le domaine de l'apprentissage automatique. C'est à travers cette plateforme que de nombreuses compétitions sont organisées.

Bien que les jeux de données disponibles soient variés et en grand nombre, notamment pour la reconnaissance d'images, il existe encore peu de matériel pour les données géospatiales. C'est notamment afin combler ce manque que des membres du département d'informatique de l'université de Louisiane et de centres de recherche de la NASA, ont constitué deux jeux de données d'imageries aéroportées : DeepSat 4 et DeepSat 6 (Basu *et al.*, 2015).

Ces données sont disponibles sur un site Internet dédié au projet DeepSat (Basu *et al.*, 2018) mais elles ont également été mises à disposition sur la plateforme Kaggle afin de favoriser leur partage. Elles sont extraites du jeu de données du programme d'imagerie national d'agriculture (National Agriculture Imagery Program, NAIP). Ce programme s'occupe de récolter de l'imagerie à 1 mètre de résolution spatiale lors du pic de la période de croissance de la végétation à l'échelle des États-Unis.

Les mosaïques du NAIP ont été découpées en tuiles de dimension 28 par 28 pixels. Cette dimension a été jugée comme un bon compris entre une taille trop petite qui diminuerait l'importance du contexte ou trop importante ce qui réduirait les propriétés statistiques de la classe (Basu *et al.*, 2015). Ces tuiles ont ensuite été classifiées manuellement en faisant attention à écarter les tuiles possédant plusieurs classes. Ainsi, les tuiles créées représentent qu'une seule classe.

Les caractéristiques des jeux de données DeepSat 4 et DeepSat 6 sont présentées dans le tableau 3. Ces jeux de données contiennent 500 000 et 405 000 images pour DeepSat 4 et DeepSat 6 respectivement.

Tableau 3 - Caractéristiques des données DeepSat (Basu et al. 2015).

	DeepSat 4	**DeepSat 6**
Nombre d'images totales	500 000	405 000
Nombre d'images d'entrainement	400 000	324 000
Nombre d'images de test	100 000	81 000
Nombre de classes	4	6
Nom des classes	Sol nu, forêt, herbe, autres	Bâtiments, sol nu, forêt, herbe, routes, eau
Résolution spatiale	1 mètre	
Taille des tuiles	28 par 28 pixels	
Formatage (profondeur des couleurs)	8 bits non-signés	
Bandes disponibles	RGB et PIR	
Lieu approximatif	État de Californie (États-Unis)	
Taille	1.36 Go	1.12 Go

Ces jeux de données permettent une certaine flexibilité dans les modèles d'apprentissage profond qui peuvent être entrainés. En effet, les images contiennent 4 bandes spectrales : rouge, vert, bleu et proche infrarouge, il est ainsi possible d'entrainer des modèles utilisant ou non certaines bandes. Par exemple, la bande proche infrarouge, qui n'est pas toujours disponible dans les images de télédétection.

Il faut noter que les images ne sont pas géoréférencées dans les jeux de données et il n'est donc pas possible de les représenter sur une carte. La seule information géographique disponible est le lieu approximatif de l'étendue géographique qui est l'État de Californie aux États-Unis.

Les données ont été téléchargées depuis le site Kaggle dans le format .MAT qui est le format propriétaire de Matlab. Les images ne sont pas stockées dans un format d'image usuel mais sous forme de matrices où les pixels de chaque bande sont codés pour prendre une valeur entre 0 et 255 inclusivement (ceci correspond à une profondeur des couleurs de 8 bits).

Le fichier Matlab contient toutes les matrices des données : les images d'entrainement, de test et les étiquettes correspondantes (tableau 4). Les matrices des images possèdent la dimension 28 par 28 pixels par nombre de classes et par nombre de bandes.

Tableau 4 - Détail des matrices dans les fichiers Matlab et leur dimension.

	Description	**DeepSat 4**	**DeepSat 6**
train_x	Images d'entrainement	28 x 28 x 4 x 400000	28 x 28 x 4 x 324000
train_y	Étiquettes des images d'entrainement	4 x 400000	6 x 324000
test_x	Images de test	28 x 28 x 4 x 100000	28 x 28 x 4 x 81000
test_y	Étiquettes des images de test	4 x 100000	6 x 81000
annotations	Nom des classes	4 x 2	6 x 2

Les étiquettes des données sont encodées dans un format *one-hot encoding*, ceci signifie qu'il y a une colonne par classe et que la classe de l'entité est indiquée par le chiffre 1, il ne peut y avoir qu'un seul 1 par entité. Par exemple, les entités du tableau 5 sont encodées sous forme de texte et les entités dans le tableau 6 sont encodées dans un format *one-hot encoding*.

Tableau 5 - Étiquettes encodées sous forme de texte.

Entité	**Classe**
1	Forêt
2	Solu nu
3	Autres

Tableau 6 - Étiquettes encodées dans un format one-hot encoding.

Entité	**Sol nu**	**Forêt**	**Herbe**	**Autres**
1	0	1	0	0
2	1	0	0	0
3	0	0	0	1

3.2.2 Données de prédiction

Les données de prédiction sont les images qui vont être classifiées. Deux images ont été choisies : Sherbrooke et Merlischachen.

La première image (figure 12) est une image de Sherbrooke qui provient de la mission WorldView-3 (DigitalGlobe, 2015).

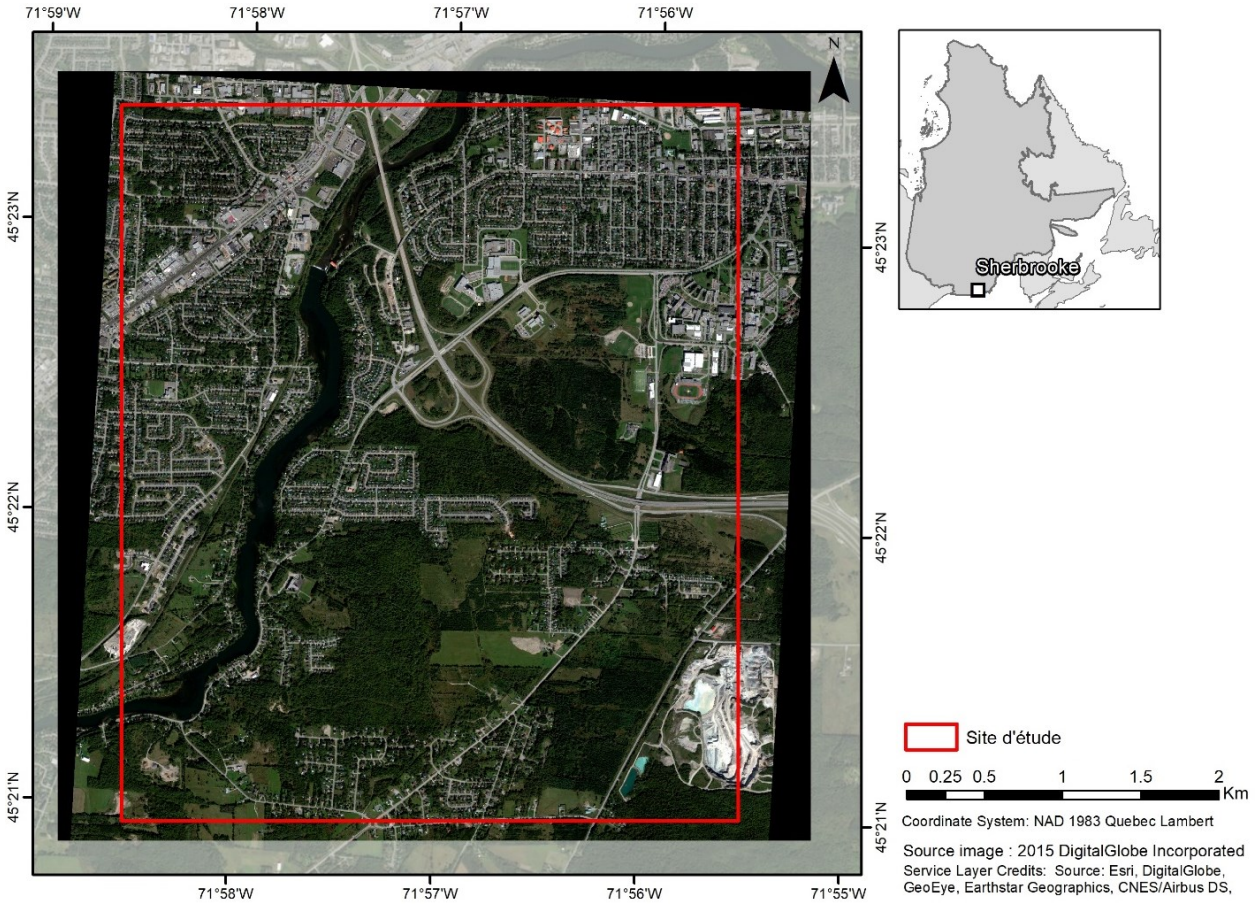


Figure 12 - Emprise de l'image de Sherbrooke et du site d'étude.

Le capteur possède une résolution spatiale de 1,24 m dans les bandes du visible et du proche infrarouge et de 0,31 m pour la bande panchromatique. Le capteur possède 8 bandes multispectrales. Seules les bandes rouge, verte, bleu et proche infrarouge ont été utilisées. Elles correspondent aux canaux 2, 3, 5 et 7 respectivement (tableau 7). La profondeur des couleurs est de 16 bits.

La bande panchromatique a uniquement été utilisée lors du prétraitement de l'image pour effectuer un *pansharpening* de l'image afin de ramener la résolution spatiale à 1 m, voir la section *Prétraitement des données de prédiction* pour le détail du traitement.

Tableau 7 - Caractéristiques du capteur à bord de WorldView-3. Uniquement les bandes utilisées dans cet essai sont montrées. (ESA, 2018).

	Nom de la bande	Longueurs d'onde	Numéro de canal	Résolution spatiale (Nadir)
Image panchromatique	Panchromatique	450 - 800 nm	-	0,31 m
Image multispectrales	Bleu	450 - 510 nm	2	1,24 m
	Vert	510 - 580 nm	3	
	Rouge	630 - 690 nm	5	
	Proche infrarouge	770 - 895 nm	7	

La deuxième image (figure 13) est une image du village **Merlischachen** qui a été acquise à l'aide d'un drone de la compagnie senseFly (senseFly 2018b). L'image provient de jeux de données destinés à l'éducation sur le site Internet de la compagnie (senseFly, 2018a).

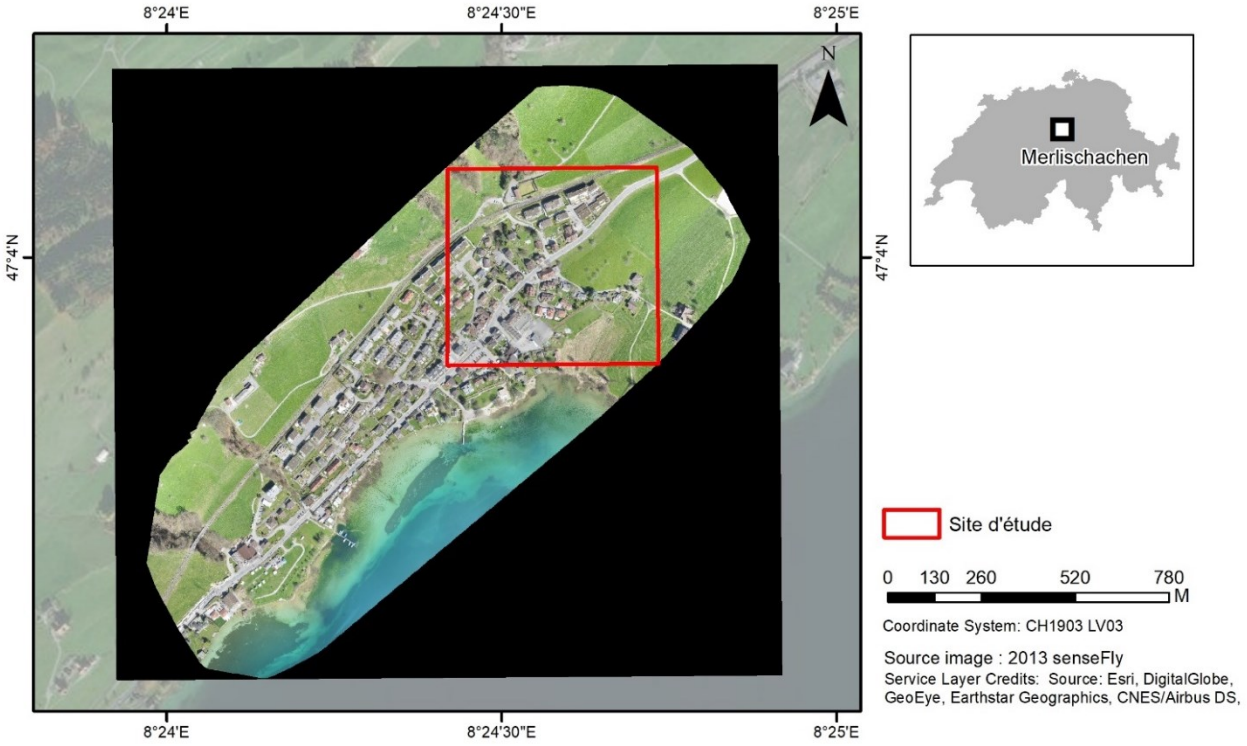


Figure 13 - Emprise de l'image de Merlischachen et du site d'étude.

L'image est une mosaïque d'images prises avec un appareil photo de type compact (Canon IXUS 125 HS). Les images ont été traitées avec le logiciel Pix4Dmapper (Pix4D, 2018) afin de créer une mosaïque orthorectifiée. Cependant, ce traitement n'a pas été effectué dans ce travail, le produit final a été téléchargé directement. La résolution spatiale originale de la mosaïque est de 5 cm.

L'image possède les bandes rouge, verte et bleu. Comme le capteur utilisé est de type commercial, il n'existe pas d'information précise sur les longueurs d'onde de chaque canal.

3.3 Méthodologie

L'approche méthodologique de cet essai est présentée à la figure 14.

La première partie de la méthodologie a été consacrée à la revue de littérature. Cette étape a notamment permis de se familiariser avec le sujet de l'apprentissage profond. C'est aussi pendant cette étape qu'une revue des principales librairies d'apprentissage profond a été constituée. Cette liste a permis d'effectuer un choix quant à la librairie à utiliser pour réaliser l'application. Ensuite, afin de mieux comprendre les interactions de la librairie d'apprentissage profond sélectionnée, un modèle conceptuel a été conçu.

Parallèlement, des jeux de données ont été sélectionnés pour servir à l'apprentissage du modèle. Des images à classifier (donnée de prédiction) ont aussi été sélectionnées puis prétraitées afin de posséder des caractéristiques similaires aux jeux de données (notamment une résolution spatiale égale). Ensuite, plusieurs modèles d'apprentissage profond ont été créés avec la librairie sélectionnée et les jeux de données.

Des modèles logique et physique de la base de données ont été créés et présentent les informations stockées et les interactions entre les entités.

Finalement, la partie implémentation de l'application a pu être effectuée. Tout d'abord, du côté serveur pour la mise en place de la base de données, la programmation de divers scripts Python, et enfin, du côté client pour la partie interface utilisateur de l'application.

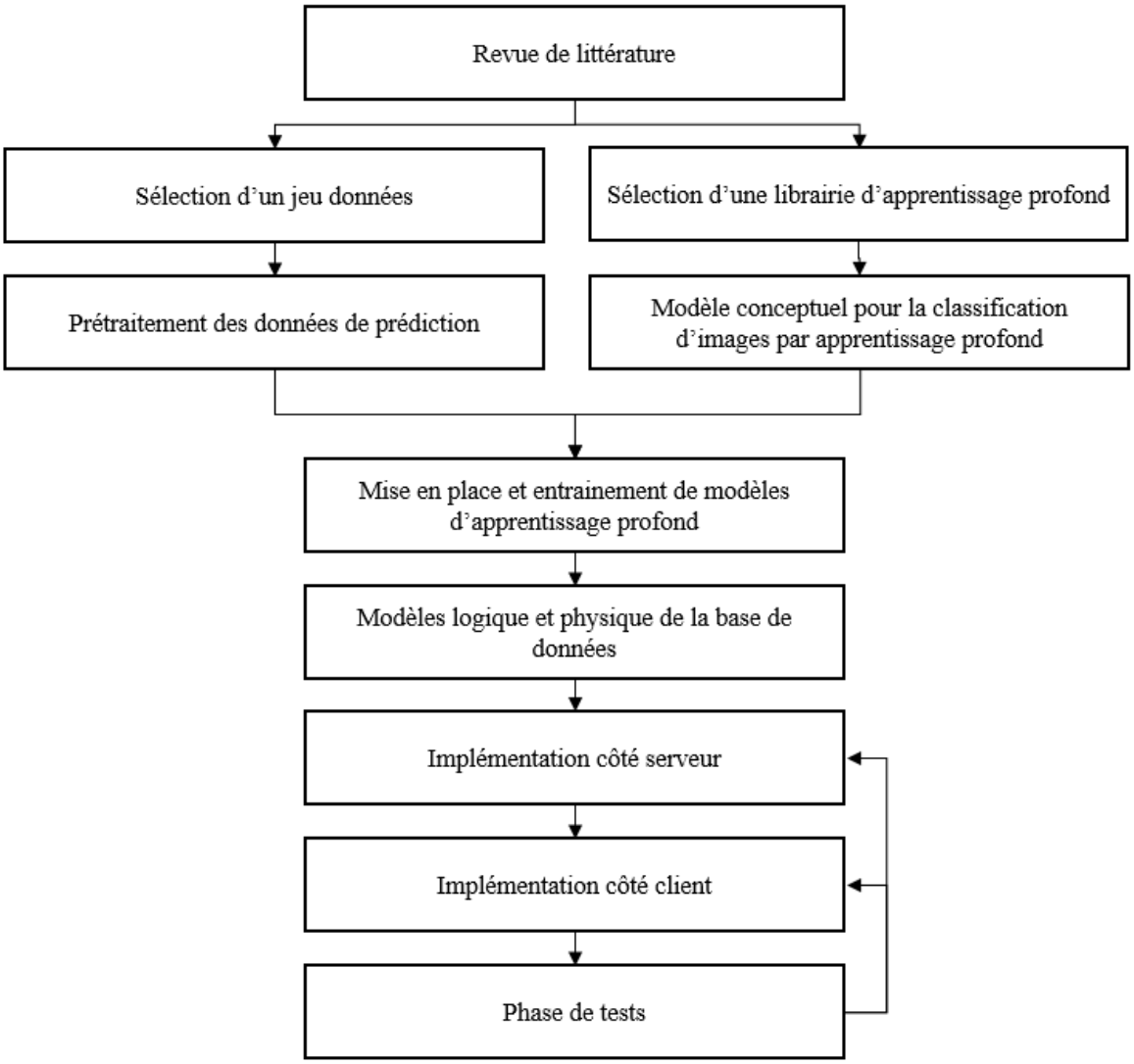


Figure 14 - Diagramme de la méthodologie.

3.3.1 Prétraitement des données de prédiction

Afin de pouvoir être classifiées, les données de prédiction doivent subir un certain nombre de prétraitements et satisfaire aux critères suivants :

- Profondeur des couleurs : 8 bits (valeurs des pixels de 0 à 255)
- Résolution spatiale : 1 mètre
- Bandes spectrales : au minimum RGB et proche infrarouge si disponible et dans l'ordre : RGB(PIR).
- Suppression des bordures : pour ne pas avoir de pixels sans valeur.

Le détail des étapes du prétraitement de l'image de Sherbrooke est le suivant :

1. **Pansharpening** : la résolution spatiale originale de l'image multispectrale étant de 1,2 m, l'image panchromatique a été utilisée pour effectuer un *pansharpening* afin de ramener la résolution de l'image multispectrale à celle de l'image panchromatique (0,31 m). La méthode de Gram-Schmidt a été appliquée avec le logiciel ArcMap (ESRI, 2015) (figure 15, à gauche). Cette méthode utilise les bandes RGB et proche infrarouge et les organise dans cet ordre.
2. **Conversion de l'image de 16 bits vers 8 bits** : en utilisant l'histogramme de chaque bande, les valeurs 16 bits non-utilisées dans les bandes ont été exclues manuellement (figure 15, à droite). Puis, l'image a été convertie de 16 bits vers 8 bits : les valeurs 16 bits minimum et maximum ont été rééchantillonnées pour devenir les valeurs 8 bits minimum et maximum.
3. **Export de l'image à 1 m de résolution spatiale** : suite au *pansharpening*, l'image avec une résolution spatiale de 0,31 m a été exportée en une image de 1 m de résolution spatiale.
4. **Découpage des bordures de l'image** : les bordures ont été exclues avec une opération de type *découpage*.

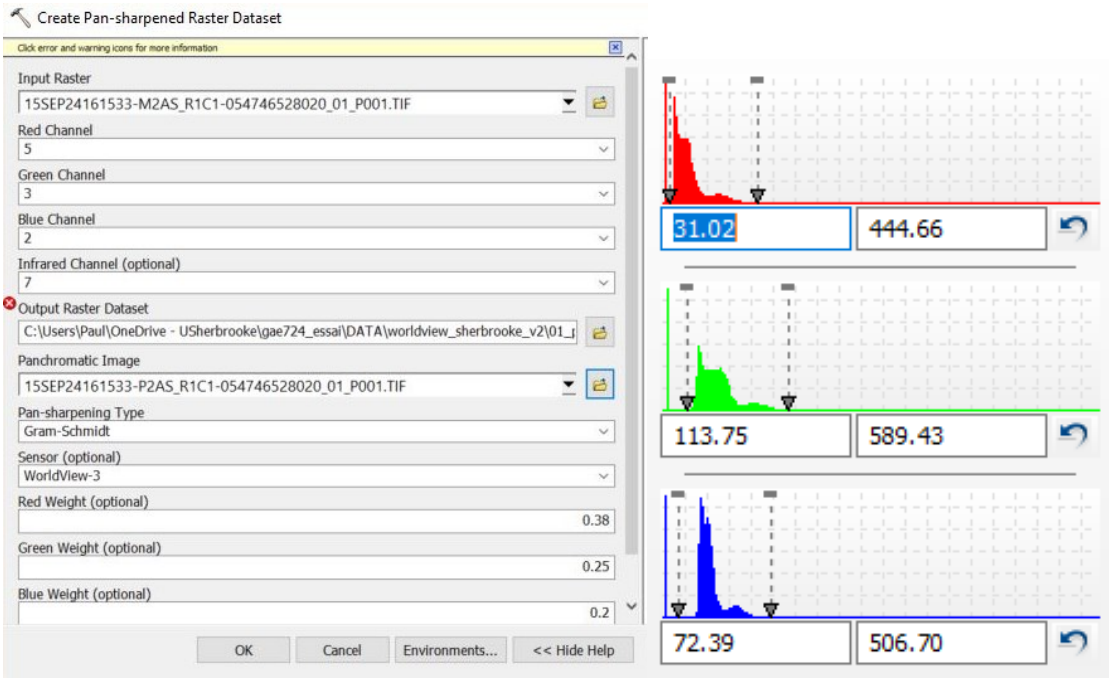


Figure 15 - À gauche, l'outil d'ArcMap utilisé pour le pansharpening (étape 1). À droite, les histogrammes des bandes RGB et les valeurs utilisées pour couper les pixels (étape 2).

L'image de Merlischachen possède déjà les bandes RGB dans le bon ordre et elle est codée 8 bits. Les étapes sont moins nombreuses :

1. **Export de l'image à 1 m de résolution spatiale** : l'image d'origine possède une résolution spatiale de 5 cm et elle a été exportée en une image de 1 m de résolution.
2. **Découpage des bordures de l'images** : l'image originale possède une forme irrégulière avec des bordures importantes, le résultat du découpage est une image beaucoup plus petite, voir la section *Données de prédiction*.

3.3.2 Sélection d'une librairie d'apprentissage profond

Le critère essentiel pour le choix de la librairie d'apprentissage profond est qu'elle doit être libre et ouverte. C'est un critère important pour assurer la pérennité de l'application car cela permet de s'affranchir de toute dépendance de renouvellement de licence dans le cas d'une librairie sous licence propriétaire. Le tableau 8 liste les librairies qui ont été considérées. Seules les solutions libres et ouvertes ont été retenues.

Tableau 8 - Comparaison de librairies d'apprentissage profond libres et ouvertes.

	TensorFlow	**Caffe**	**Keras**	**PyTorch**
Prise en main	+	+	+++	++
Documentation	++	++	++	++
Flexibilité	+++	++	++	+++

Les librairies ont été comparées sur la base de trois critères :

- Prise en main : courbe d'apprentissage, facilité pour se familiariser avec la librairie.
- Documentation : qualité et disponibilité de la documentation (exemples, tutoriels, etc.)
- Flexibilité : types d'architectures supportés, opérations possibles, etc.

La facilité de prise en main est le critère le plus important parmi les trois critères car le temps limité de l'essai ne permet pas de passer beaucoup de temps à apprendre à utiliser une librairie. Keras est la librairie qui l'emporte pour ce critère, suivie de PyTorch et finalement TensorFlow et Caffe à égalité. Keras présente l'API la plus simple et intuitive pour débuter en apprentissage profond.

Concernant, la documentation, les librairies arrivent toutes à égalité. Elles disposent toutes de nombreuses ressources, tutoriels et exemples sur Internet. Il y a néanmoins une petite préférence pour Keras avec le livre *Deep Learning with Python* (Chollet, 2018). Ce livre est un bon guide pour débuter en apprentissage profond de façon générale et il est basé sur la librairie Keras.

Finalement, pour le critère de la flexibilité, TensorFlow et PyTorch l'emportent à égalité, suivi de Keras et de Caffe. TensorFlow et PyTorch permettent de manipuler les tenseurs à un niveau assez bas et ainsi de créer des réseaux plus complexes. Keras est une API de haut niveau et est donc plus limitée, néanmoins, il est possible de choisir entre plusieurs librairies comme moteur en arrière-plan comme TensorFLow, Theano ou CNTK. Caffe perd un point car elle est dédiée à l'analyse d'images et non de texte ou de parole (bien que ces thèmes soient moins couramment utilisés en géomatique).

Le choix s'est finalement porté sur la librairie Keras. C'est une librairie bien adaptée pour démarrer un apprentissage profond car elle est simple à manipuler mais son potentiel n'est pas pour autant limité.

3.3.3 Modèle conceptuel pour la classification d'images par apprentissage profond

Le modèle conceptuel pour la classification d'images par apprentissage profond est présenté à la figure 16. Il montre les étapes et leurs interactions pour entrainer un modèle d'apprentissage profond pour classifier des images.

Les images classifiées (dont les classes sont connues), doivent d'abord être formatées afin d'être dans un format compatible avec le modèle d'apprentissage profond. Essentiellement, elles doivent être formatées dans des tenseurs, voir la section *Fonctionnement des réseaux de neurones artificiels*. Le détail des prétraitements est présenté dans la section *Formatage des données* ci-après.

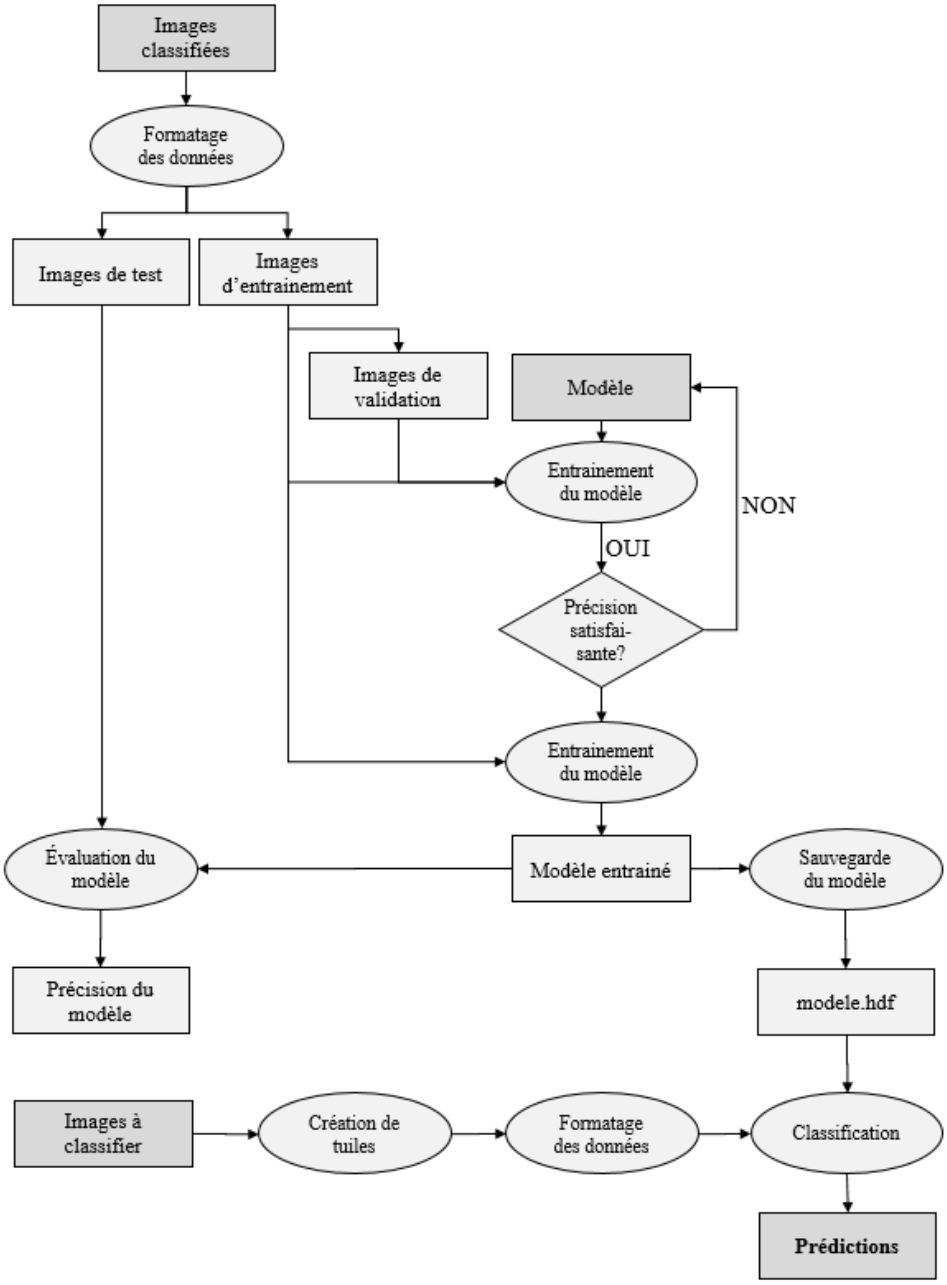


Figure 16 - Modèle conceptuel pour la classification d'images par apprentissage profond.

Une fois les données formatées, elles sont divisées en deux groupes : les images d'entrainement et les images de test. À noter que les jeux de données utilisés dans cet essai (DeepSat 4 et 6) sont déjà divisés dans ces deux groupes. Ensuite, une partie des images d'entrainement sont mises de côté pour servir d'images de validation, se référer à la section *Fonctionnement des réseaux de neurones artificiels* dans le cadre théorique pour plus d'explications sur la nécessité d'effectuer ces

séparations. À noter qu'avec Keras, il n'est pas nécessaire de séparer effectivement les images d'entrainement, cette séparation se fait lors de l'entrainement du modèle en indiquant un pourcentage.

L'architecture du modèle est définie dans l'entité « Modèle » du schéma. L'entraiment du modèle est effectué après et est une étape qui nécessite d'être répétée plusieurs fois jusqu'à obtenir une précision jugée suffisante. À chaque itération, l'architecture du modèle est modifiée un peu jusqu'à obtenir une précision adéquate. Le détail des architectures utilisées dans cet essai est présenté dans la section *Architectures des modèles d'apprentissage profond*.

Une fois que la précision est jugée suffisante, le modèle est à nouveau entrainé mais avec toutes les données d'entrainement (entrainement et validation). C'est seulement ensuite que la précision du modèle peut être évaluée avec des données que le modèle n'a jamais vues (images de test).

Avec Keras, il est possible de sauvegarder le modèle dans un format .HDF, les poids ajustés sont notamment stockés dans ce fichier, ceci afin que le modèle puisse être utilisé par la suite sans devoir effectuer à nouveau l'entraiment du modèle qui est une étape qui demande beaucoup de ressources de calcul.

Les images à classifier doivent d'abord être découpées en tuiles de la même dimension que les images classifiées, puis formatées sous forme de tenseurs. Finalement, il est enfin possible d'appliquer le modèle entrainé pour classifier ces tuiles. Le résultat est un score d'appartenance à chaque classe pour chaque tuile. Similairement à une probabilité, la somme des scores est égale à 1.

3.3.3.1 Formatage des données

Les images classifiées doivent être formatées sous forme de tenseurs pour servir à l'entrainement et à l'évaluation des modèles. Les étapes suivantes ont été effectuées dans un script Python :

1. **Lecture du fichier .MAT** : le fichier .MAT est un format propriétaire, cependant il est possible de le lire avec la librairie libre et ouverte Scipy.
2. **Redimensionnement des matrices** : les matrices (tenseurs) du fichier .MAT étant stockées dans le format [hauteur, largeur, bandes, entité], il a fallu les reformater pour satisfaire la convention utilisée par TensorFlow, à savoir : [entité, largeur, hauteur, bande].

3. **Sauvegarde des matrices au format NumPy** : les tenseur redimensionnés ont été stockés dans le format .NPY de NumPy qui est plus facile à manipuler par la suite.

3.3.3.2 Architectures des modèles d'apprentissage profond

La définition de l'architecture d'un modèle est faite de manière assez intuitive avec Keras. Il suffit d'ajouter des couches l'une après l'autre à un modèle. La difficulté est de savoir quelle succession de couches et quels paramètres doivent être utilisés. Plusieurs itérations sont souvent nécessaires avant de trouver un modèle qui donne de bons résultats.

Afin de pouvoir classifier des images avec et sans la bande proche infrarouge avec l'une ou l'autre des deux bases de données, quatre modèles au total ont été entrainés :

- Deepsat4-3bands
- Deepsat4-4bands
- Deepsat6-3bands
- Deepsat6-4bands

Les modèles ont été entrainés avec la même architecture de modèle à quelques différences près selon le nombre de bandes utilisées et le nombre de classes. En effet, les images étant assez semblables dans les jeux de données, la précision obtenue était satisfaisante en procédant de la sorte. L'architecture de modèle a ainsi été développée avec le jeu de données de DeepSat 6 avec 4 bandes puis appliquée à la version avec 3 bandes puis à DeepSat 4 avec 4 et 3 bandes.

Le modèle développé dans cet essai est un réseau de neurones convolutif d'un problème de type multi-classes vers une seule classe. En effet, il y a plusieurs classes possibles et chaque image appartient à une seule classe dans les jeux de données. Ainsi, la fonction de perte utilisée est la fonction *categorical_crossentropy* (tableau 1). Le modèle est inspiré d'un modèle convolutif de Chollet (2018) et d'un modèle disponible sur Kaggle (Adivarekar, 2018). Le résumé du modèle est montré dans la figure 17 et le script complet qui a servi à entrainer le modèle est disponible en *Annexe 1- Script Python entrainement du modèle DeepSat 6 avec 4 bandes*.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        1184
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390
=================================================================
Total params: 93,926
Trainable params: 93,926
Non-trainable params: 0
_________________________________________________________________
```

Figure 17 - Résumé de l'architecture du modèle DeepSat 6 avec 4 bandes.

Essentiellement, le modèle contient une succession de couches de convolution 2D et de *pooling* qui sont typiquement utilisées dans les réseaux de neurones convolutifs. La couche *Flatten* est utilisée pour reformatter les données et la couche *Dropout* permet de régulariser le surapprentissage. La dernière couche du modèle est une couche de type *Dense* avec en sortie 6 paramètres, puisqu'il y a 6 classes possibles avec DeepSat 6. C'est donc une des couches à modifier lorsque le modèle est entrainé avec le jeu de données DeepSat 4 puisqu'il y a 4 classes possibles.

3.3.4 Modèle conceptuel des données

Le modèle conceptuel des données est présenté à la figure 18. Ce modèle montre les entités qui seront stockées dans la base de données, leurs informations ainsi que les relations entre celles-ci.

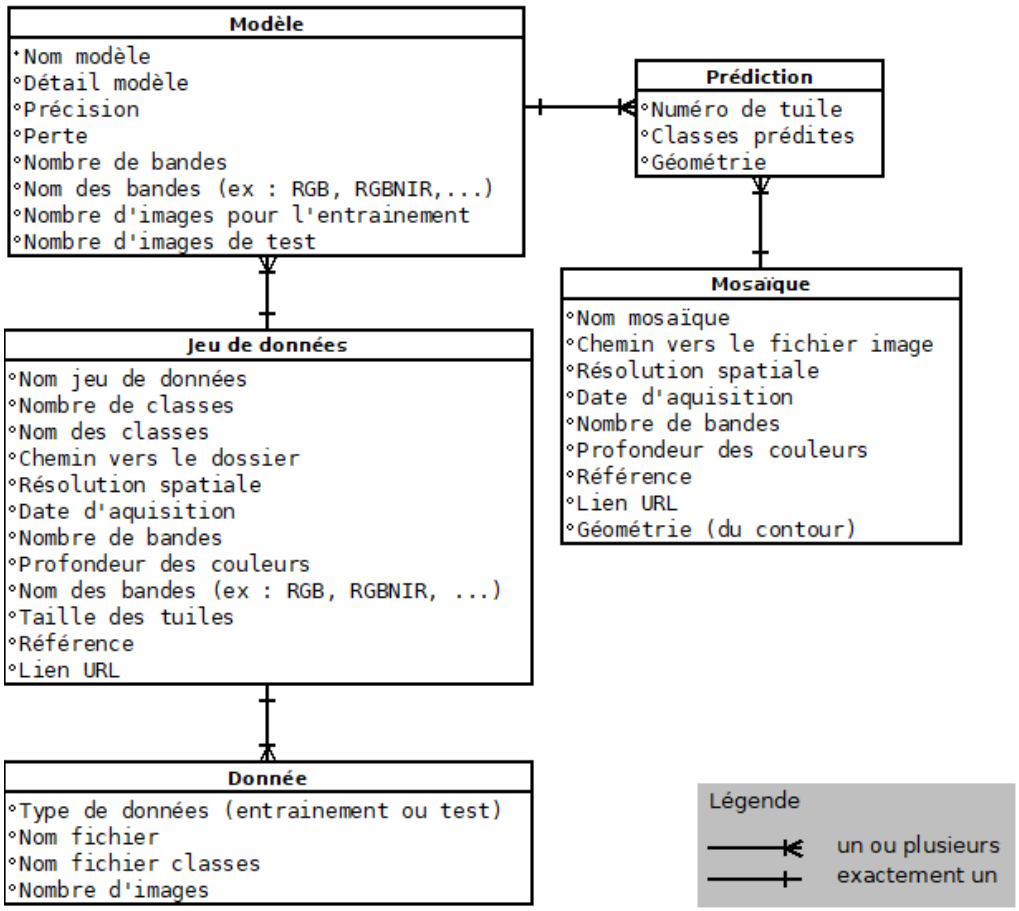


Figure 18 - Modèle conceptuel des données.

L'entité **Jeu de données** contient les informations sur les jeux de données disponibles dans la base de données. On y retrouve notamment le nombre de classes et leurs noms (par exemple : sol nu, forêt, bâtiments, etc.). Elle contient également des informations de télédétection comme la résolution spatiale, la profondeur des couleurs (par exemple, 8 bits) ou encore le nombre et le nom des bandes des images. Finalement, cette entité contient quelques informations sur la provenance des données dans les attributs « Référence » et « Lien URL ».

À chaque jeu de données de cette table sont associés deux données dans l'entité **Données**, une pour les données d'entrainement et une pour les données de test. Cette information est stockée dans l'attribut « Type de données ».

Pour des questions de performance de la base de données, les jeux de données pouvant être très volumineux, les données ne sont pas stockées directement dans la base de données mais dans des fichiers sur le serveur. Chaque donnée pointe vers deux fichiers de type NumPy : l'un contenant les images sous forme de tenseur et l'autre les classes associées aux images. Le chemin vers le

fichier est composé par l'attribut « Chemin vers le dossier » de l'entité Jeu de données et « Nom fichier » pour les images ou « Nom fichier classes » pour les classes de l'entité Données. Ainsi, pour un jeu de données les fichiers suivants sont stockés sur le serveur :

- Données d'entrainement
    - Images d'entrainement
    - Classes associées à chaque image d'entrainement
- Données de test
    - Images de test
    - Classes associées à chaque image de test

L'entité **Modèle** contient les informations sur les modèles disponibles dans la base de données. Ces modèles sont entrainés à partir d'un jeu de données. On trouve dans l'attribut « Détail modèle » l'architecture détaillée du modèle, notamment le type de couche et le nombre de paramètres. Le nombre de bandes utilisées est également une information qui figure dans cette entité puisqu'il est possible d'entrainer un modèle en utilisant moins de bandes que disponibles dans les images d'entrainement.

L'entité **Mosaïque** contient les informations sur les images à classifier de la base de données. Similaire à l'entité Jeu de données, elle comporte des informations de télédétection comme la résolution spatiale, la profondeur des couleurs, le nombre de bandes, etc. mais également le géoréférencement ou l'emprise du contour de l'image. Cette information est utilisée pour l'affichage de l'image dans l'interface Web. L'image n'est pas directement stockée dans la base de données mais plutôt son chemin d'accès avec l'attribut « Chemin vers le fichier image ».

Les prédictions de classes étant effectuées par tuile, l'image à classifier doit d'abord être divisée en tuiles de même dimension avant de pouvoir faire des prédictions. Ces tuiles sont stockées dans l'entité **Prédiction**. La cardinalité du schéma exprime qu'un modèle peut être appliqué à plusieurs images et qu'une image peut être classifiée avec différents modèles. En revanche, il n'est pas possible classifier plusieurs fois la même image avec le même modèle. Ainsi, chaque tuile de l'entité Prédiction correspond à une seule image et à un seul modèle.

L'attribut « Classes prédites » contient la prédiction de la classification.

3.3.5 Modèle physique des données

Le modèle physique des données est présenté à la figure 19. Il exprime le modèle conceptuel des données sous forme de tables et de colonnes tel qu'implémenté dans la base de données. Les termes ont été traduits en anglais pour obtenir des noms de colonnes plus courts.

À noter que pour une définition précise de chaque attribut, un dictionnaire des données est disponible en *Annexe 2 - Dictionnaire des données du modèle physique*.

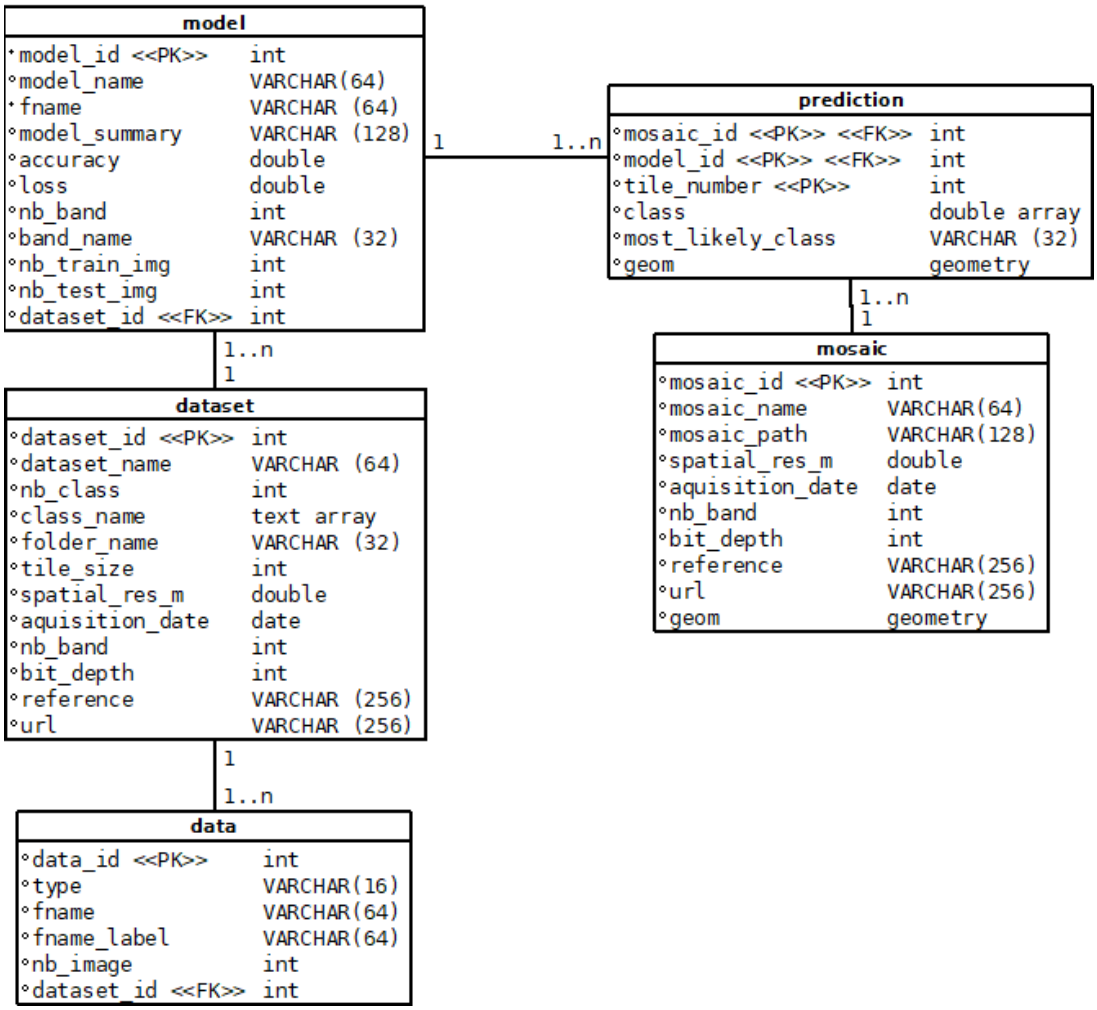


Figure 19 - Modèle physique des données

Les tables du modèle physique correspondent aux entités identifiées dans le modèle conceptuel. Également, on retrouve les attributs du modèle conceptuel sous forme de colonnes avec l'addition de certaines colonnes, notamment, des clés primaires qui servent d'identifiants uniques pour chaque table et des clés étrangères pour exprimer les relations entre les tables dans la base de données.

Concernant les clés des tables, on peut noter une petite particularité pour la table « prediction » dont la clé primaire est composée de trois clés primaires dont deux étrangères qui font référence soit à la table « mosaic » soit à la table « model ». En effet, comme mentionné plus haut pour le modèle conceptuel, les prédictions sont obtenues en appliquant un modèle à une image. Ainsi, par exemple, pour un modèle X et une image A, il ne peut y avoir qu'une seule prédiction pour chaque tuile. En effet, il ne sert à rien d'appliquer plusieurs fois le modèle X à l'image A puisque le résultat de la classification sera le même. Les clés d'identification de cette table permettent ainsi d'éviter une duplication d'entités.

La table « prediction » contient une colonne supplémentaire : « most_likely_class ». Cette colonne prend le nom de la classe la plus probable selon les scores de la colonne « class ». Cette colonne est notamment utilisée pour la symbologie de la couche dans l'application Web.

3.3.6 Technologies utilisées

L'implémentation de l'application Web fait appel à plusieurs technologies de programmation qui sont illustrées à la figure 20.

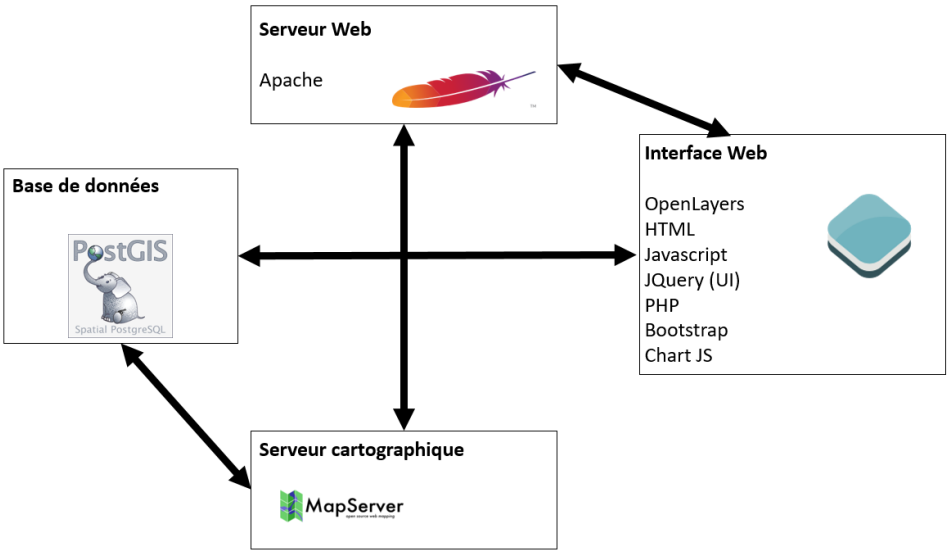


Figure 20 - Schéma des technologies utilisées et leurs interactions.

Pour le stockage, il existe plusieurs systèmes de gestion de base de données (Oracle, MySQL, SpatialLite, etc.) mais c'est PostgreSQL avec l'extension spatiale PostGIS qui a été choisie. C'est une solution libre et ouverte largement utilisée et qui est bien intégrée avec la plupart des autres technologies utilisées dans cet essai.

Du côté du serveur cartographique, il existe principalement deux solutions libres et ouvertes : GeoServer et MapServer. GeoServer est une solution plus conviviale alors que MapServer est réputée plus fiable et rapide. L'une ou l'autre de ces solutions auraient pu être utilisées et c'est MapServer qui a été retenue.

Concernant l'interface de programmation Web, les technologies suivantes ont été utilisées :

- OpenLayers : librairie JavaScript pour la cartographie en ligne.
- JQuery : librairie utilisée pour faciliter la programmation en JavaScript.
- JQuery (UI) : librairie graphique en JQuery.
- HTML/JavaScript : langages de mise en forme et d'interaction avec le navigateur Web.
- CSS : langage pour la gestion des styles de l'interface.
- Bootstrap : librairie et outils permettant notamment de concevoir une interface adaptative (indépendante de la taille de l'écran).
- PHP : langage côté serveur permettant d'interagir avec la base de données et de lancer des scripts Python.

Par ailleurs, le serveur fonctionne sous Debian 6 avec Apache 2 pour serveur Web.

Le langage Python a été utilisé pour la manipulation des données, les géotraitements et pour l'élaboration et l'entrainement de modèle d'apprentissage profond. C'est également le langage utilisé pour faire les traitements à la volée dans l'application. Un nombre varié de librairie libres et ouvertes ont été utilisées et sont présentées dans le tableau 9.

Tableau 9 - Librairies Python utilisées.

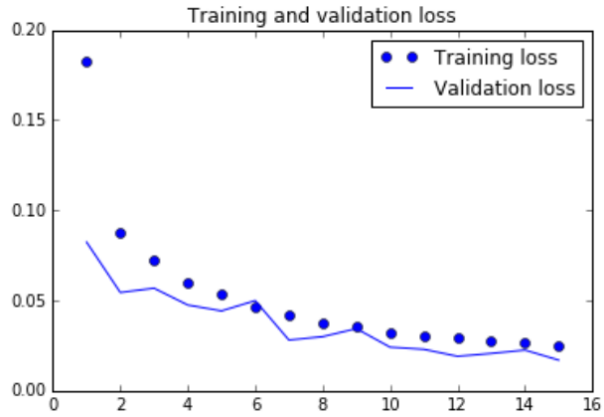
Librairies	Commentaires
Keras	Librairie d'apprentissage profond
TensorFlow	Moteur en arrière-plan utilisé avec Keras
NumPy	Manipulation de matrices et tenseurs
Matplotlib	Affichage et création de graphes
Psycopg2	Manipulation de la base de données PostGIS
OGR/GDAL	Lecture et écriture de données vectorielles et matricielles
OSR	Gestion des systèmes de référence spatiaux

4. Résultats

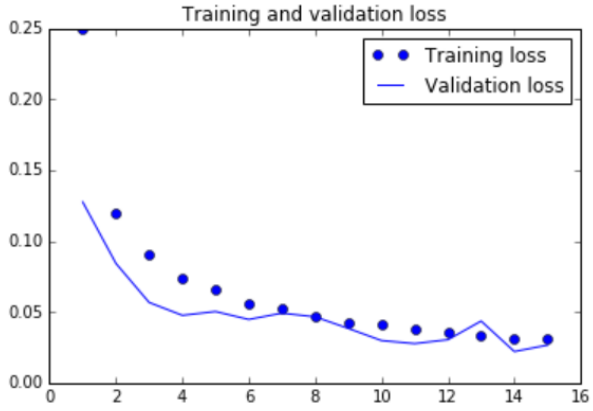
4.1 Entrainement des modèles

La figure 21 montre l'évolution des pertes des données d'entrainement et de validation pour chaque modèle en fonction des époques. On voit que cette valeur diminue pour les données d'entrainement et de validation pour atteindre environ entre 0,1 et 0,3 après 15 époques.

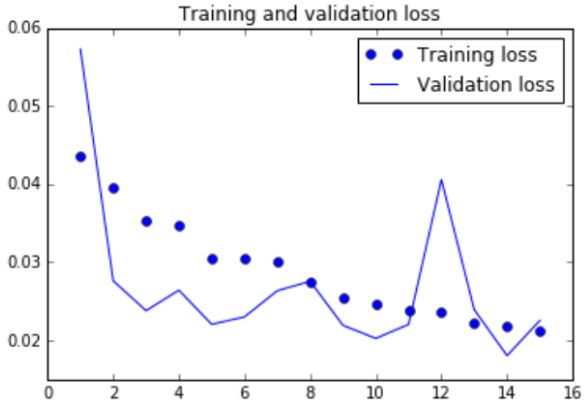
Avec les 324 000 images de DeepSat 6, l'entrainement a duré environ 4 à 5 minutes avec une carte graphique Nvidia GeForce 1060 et 6 Go de RAM.



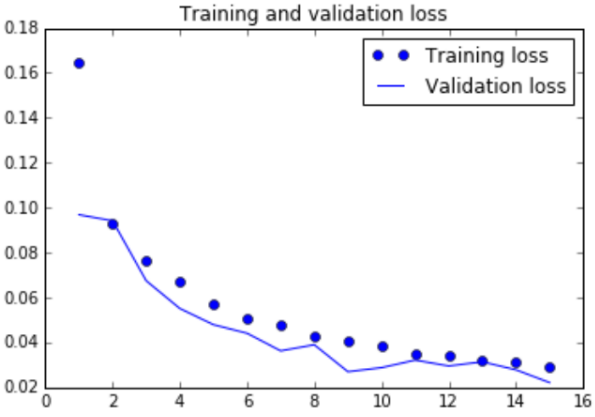
DeepSat 4 – 4 bandes



DeepSat 4 – 3 bandes



DeepSat 6 – 4 bandes



DeepSat 6 – 3 bandes

Figure 21 - Valeurs de perte des données d'entrainement et de validation par époques.

Le tableau 10 montre les valeurs de précision et de perte pour chaque modèle après l'évaluation. On voit que la précision est toujours au-dessus de 99%. Par ailleurs, la perte est comprise entre 0,010 et 0,025.

Tableau 10 - Valeurs de précision et de perte des modèles après évaluation.

	Précision	**Perte**
DeepSat 4 – 4 bandes	0,99644	0,01021
DeepSat 4 – 3 bandes	0,99407	0,01750
DeepSat 6 – 4 bandes	0,99602	0,01176
DeepSat 6 – 3 bandes	0,99233	0,02467

4.2 Présentation de l'application et de son fonctionnement

L'interface utilisateur de l'application est présentée à la figure 22. Elle est composée de deux éléments principaux :

- la carte de fond, qui occupe la majeure partie de la fenêtre pour faciliter l'exploration de la carte;
- et un panneau sur la gauche, qui permet d'accéder aux fonctionnalités de l'application et d'interagir avec la carte.

Les options de contrôle de la carte sont celles par défaut d'OpenLayers. Ainsi, les outils de navigation sont basiques : le zoom se fait, soit à l'aide de la molette de la souris, soit en cliquant sur les boutons plus et moins du coin supérieur droit, quant au déplacement, il se fait de manière conventionnelle en cliquant à n'importe quel endroit de la carte et en glissant la souris.

Il est possible de changer le type de fond de carte avec les boutons *OSM* et *Satellite* situés en-haut du panneau de gauche. Le fond de carte de type route est la carte standard l'OpenStreetMap et le fond de carte satellite est celui des images de Bing Maps. L'information sur la source des données des fonds de carte se trouve dans le coin inférieur droit en cliquant sur le bouton représenté par la lettre « i ».

Figure 22 - Interface utilisateur de l'application.

Le panneau de gauche contient les fonctionnalités avancées de l'application et constitue l'effort principal de ce travail. Ces fonctionnalités des sections du panneau de gauche sont décrites dans les sous-sections suivantes.

Les menus déroulants ainsi que les boutons du panneau sont générés dynamiquement en interrogeant la base de données du serveur. Les technologies utilisées reposent sur l'utilisation de scripts PHP et le rechargement asynchronisé d'une partie de la page Web avec AJAX.

4.2.1 Ajouter une image

La première section du panneau de gauche est une fonctionnalité qu'il n'a malheureusement pas été possible de développer dans le cadre de cet essai par manque de temps. Ainsi, le bouton *Parcourir…* est grisé et il est impossible de cliquer dessus.

Cette fonctionnalité devra permettre de télécharger une image géoréférencée sur le serveur. Cette image s'affichera ensuite sur le fond de carte et pourra être classifiée à l'aide d'un modèle d'apprentissage profond sélectionné dans la section suivante du panneau.

Cette fonctionnalité ne représente pas un défi technique majeur mais dans le temps restreint de cet essai, il a été préféré de se concentrer sur les aspects novateurs de l'application. Cette fonctionnalité est néanmoins importante et fait partie de l'interface utilisateur pour apporter plus de réalisme à l'application (figure 23).

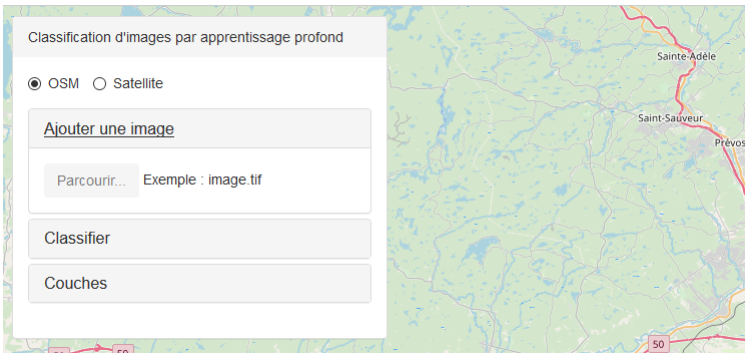


Figure 23 - Capture d'écran de la première section du panneau de gauche de l'application permettant d'ajouter une image.

4.2.2 Classifier une image

La deuxième section du panneau de gauche permet de classifier une image avec un modèle d'apprentissage profond (figure 24).

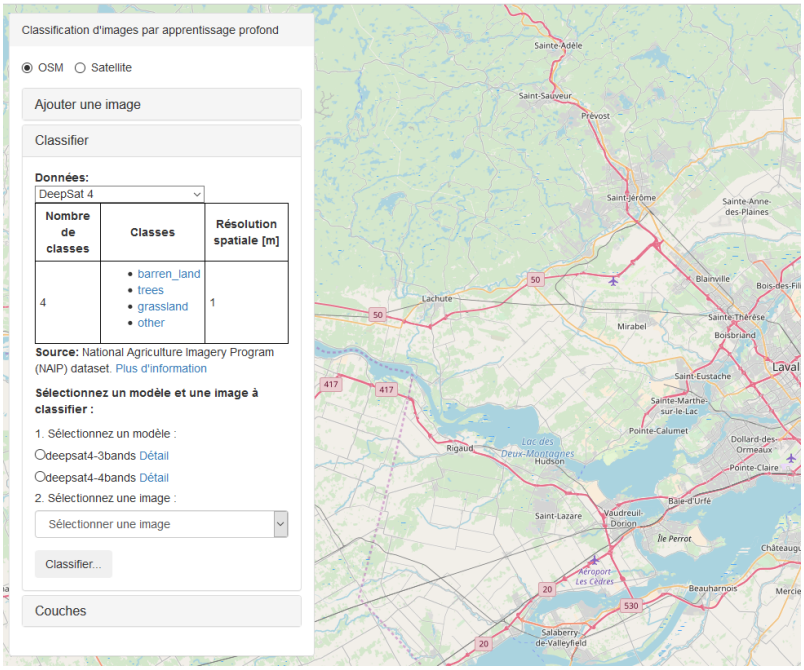


Figure 24 - Capture d'écran de la deuxième section du panneau de gauche permettant de classifier une image.

Afin de sélectionner le modèle le plus approprié pour classifier l'image, il est possible d'explorer les jeux de données présents sur le serveur. Présentement, il y a deux jeux de données : DeepSat 4 et DeepSat 6.

Le menu déroulant permet de sélectionner une base de données afin d'afficher les informations suivantes :

- Le nombre et le nom des classes
- La résolution spatiale des images
- La source des données. Il est possible d'accéder au site internet des données en cliquant sur le lien *Plus d'information*.

Pour chacune des classes listées, il est possible de cliquer dessus afin d'afficher un échantillon d'images de cette classe (figure 25). Lorsqu'une classe est cliquée, une fenêtre s'ouvre et affiche 9 échantillons disposés sur une grille de 3 par 3 d'images sélectionnées aléatoirement. Le nombre d'images de cette classe s'affiche également dans la fenêtre.

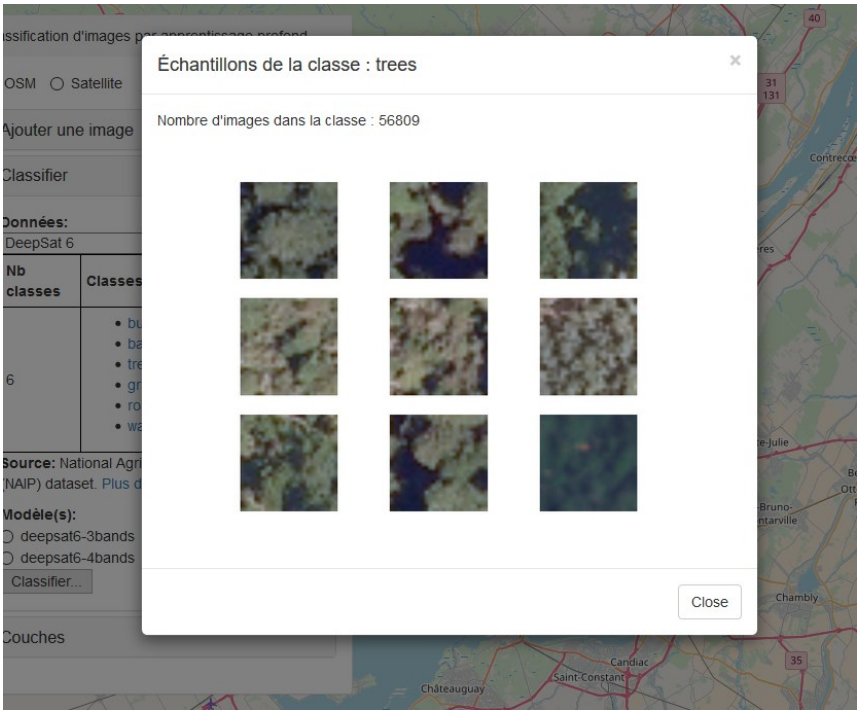


Figure 25 - Capture d'écran d'un échantillon de la classe « Forêt ».

Techniquement, lorsqu'une classe est cliquée, un script Python est lancé sur le serveur par une ligne de commande effectuée en PHP. Le script Python charge les images d'entrainement (train_x) ainsi que les étiquettes associées (train_y) dans la RAM du serveur. Celles-ci sont stockées dans des matrices de type NumPy sur le serveur. Cette opération est relativement lente car les bases de données font plus de 1 Go.

Ensuite, les images de la classe demandée sont extraites du jeu de données, puis mélangées aléatoirement. Finalement, les neuf premières images sont écrites dans un seul fichier image à l'aide de la librairie Python Matplotlib. C'est cette image qui est affichée dans la fenêtre.

Lorsqu'un jeu de données est sélectionné, les modèles correspondant sont listés sous forme de boutons en-dessous des informations de celui-ci.

Pour chacun des deux jeux de données présents sur le serveur, deux modèles ont été entrainés :

- un avec les images de 4 bandes (RGB et proche infrarouge) ;
- et un avec les mêmes images mais possédant seulement 3 bandes (RGB).

Plus d'information sur chacun des modèles peut être obtenue en cliquant sur *Détail*. L'architecture ainsi que la précision, la perte et le nom des bandes du modèle s'affichent dans une fenêtre.

Finalement, un menu déroulant permet de sélectionner une des images présente sur le serveur à classifier. Le bouton *Classifier…* permet de lancer la classification de l'image en utilisant le modèle sélectionné. Lorsque le bouton est cliqué, un script Python est lancé sur le serveur à l'aide d'une ligne de commande effectuée en PHP. Ce script est assez lourd et demande beaucoup de temps surtout pour une image très étendue.

Les étapes principales de ce script sont les suivantes :

1. Lecture de l'image sélectionnée avec la librairie GDAL.
2. Découpage de l'image en tuiles de mêmes tailles que les images du jeu de données sélectionné.
3. Formatage des tuiles en tenseur pour la classification avec le modèle d'apprentissage profond.
4. Classification des tuiles avec le modèle sélectionné.

5. Ajout des tuiles dans la base de données sous format vectoriel ainsi que la classification.
6. Calcul d'une classe la plus probable en fonction de la prédiction.

L'étape 2 du découpage des tuiles se connecte d'abord à la base de données afin de récupérer la taille des tuiles dans la table *database*, puis elle va découper l'image en parcourant les pixels en partant du coin supérieur gauche. Si le nombre de pixels dans la largeur de l'image à classifier n'est pas un multiple de la taille des tuiles, les pixels excédentaires seront ignorés par le découpage. Ainsi, il se peut que les tuiles créées ne couvrent pas totalement l'image dans sa largeur. La même remarque est valable pour la hauteur de l'image. Les pixels excédentaires dans la hauteur seront également ignorés par le découpage. C'est pour cela que la classification ne couvre pas forcément totalement l'image.

Également, lors de l'étape du découpage, un tenseur contenant les valeurs des pixels dans chaque bande pour chaque tuile est créé. La dimension de ce tenseur est [nombre de tuiles, hauteur des tuiles, largeur des tuiles, nombre de bandes]. Cette étape permet ainsi d'obtenir les données dans un format adéquat pour la classification par le modèle d'apprentissage profond.

L'étape 3 du formatage des tuiles va formater les valeurs des pixels contenues dans le tenseur créé précédemment. Cette étape consiste à :

1. Couper la valeur des pixels dans l'intervalle des valeurs possibles en fonction de la profondeur des couleurs (0 à 255 pour 8 bits). Cette étape permet de gérer les pixels sans valeur potentiellement présents dans l'image comme par exemple dans le cas de bordures. Les valeurs au-dessous de zéro sont affectées à zéro et les valeurs au-dessus de la valeur maximale sont affectées à la valeur maximale.
2. Diviser les pixels par la valeur maximale de la profondeur des couleurs (255 pour 8 bits). Cette étape crée des valeurs plus petites à manipuler pour la librairie Keras, ce qui est recommandé.

Néanmoins, il faut préciser que la librairie d'apprentissage profond Keras n'a malheureusement pas pu être installée sur le serveur à cause d'une version Python désuète sur le serveur (Python 2.6). Ainsi, la librairie Keras n'est pas directement utilisée dans l'étape de la classification et une alternative a dû être trouvée : les images ont été classifiées préalablement sur un autre ordinateur avec Keras.

Plus précisément, les images ont d'abord été découpées en tuiles puis enregistrées dans des tenseurs au format NumPy. Ensuite, ces tenseurs ont été classifiés avec différents modèles (voir le tableau 11 pour la liste des classifications effectuées). Finalement, le résultat de la classification, qui se présente dans la forme d'une matrice, a été sauvegardé au format NumPy. C'est cette matrice qui est stockée sur le serveur et qui est utilisée pour associer la prédiction à la tuile dans le script Python. Il y a ainsi une matrice de prédiction par modèle et par image.

Tableau 11 - Classifications effectuées des images et modèles.

Images	Modèles
Merlischachen	Deepsat4-3bands Deepsat6-3bands
Sherbrooke	Deepsat4-4bands Deepsat6-4bands

Le script Python a été développé avec la possibilité d'utiliser la librairie Keras dans le cas où l'application est déplacée sur un serveur possédant Keras. Les lignes correspondantes ont été mises en commentaires. Ainsi, il suffira d'ajouter ces lignes pour adapter le script à un serveur avec Keras.

L'étape 6 va remplir la colonne « most_likely_class » de la table « prediction » en utilisant les scores de la prédiction. Les scores sont triés par ordre croissant et, si la différence entre le premier et le deuxième score est plus grande que deux fois la moyenne, la classe la plus probable est la classe associée au premier score, sinon la classe est « unknown » : indéterminée.

4.2.3 Explorer les résultats de la classification

Finalement, la troisième section du panneau, permet d'explorer les résultats de la classification sur la carte de fond (figure 26). En sélectionnant une image dans le menu, les deux actions suivantes sont déclenchées :

1. Centrage de la carte sur l'image sélectionnée.
2. Listage des modèles utilisés pour classifier l'image sous forme de boutons.

Figure 26 - Troisième section du panneau de gauche permettant d'explorer les résultats de la classification.

Le bouton *Masquer* permet de masquer l'image de la carte de fond.

En cliquant sur un des modèles listés, la classification de l'image s'affiche sur la carte. En fonction de la taille de l'image, ou plus précisément du nombre de tuiles, l'affichage peut prendre quelques secondes. En effet, lorsque qu'un modèle est cliqué, une requête SQL est lancée par un script PHP. Cette requête crée une vue (ou table virtuelle) dans la base de données. Cette vue sélectionne les colonnes suivantes de la table *prediction* :

- *tile_number* : afin d'avoir un identifiant unique pour la table.
- *class* : cette colonne est utilisée afin de créer une colonne par classe. Pour rappel, cette colonne est une liste de taille variable qui contient une variable par classe. La requête SQL est créée dynamiquement afin de découper cette colonne en une colonne par classe. Le nom des colonnes est récupéré de la table *database* pour correspondre avec les valeurs de prédictions, figure 27.
- *most_likely_class* : colonne de la classe la plus probable.
- *geom* : géométrie de la tuile.

tile_number	barren_land	trees	grassland	other	most_likely_class	geom
0	0	0.9996	0.0003	0	trees	01030000A0E610000001000000050000012747F7E59D0204...
1	0	0.9996	0	0.0003	trees	01030000A0E610000001000000050000000B8096D389D0204...
2	0	0	0.9999	0	grassland	01030000A0E610000001000000050000007DFAD28BAD0204...
3	0	0.9451	0.0548	0	trees	01030000A0E610000001000000050000F690C57DEAD0204...
4	0	1	0	0	trees	01030000A0E610000001000000050000CC95DDD21AD1204...
5	0	0.9999	0	0	trees	01030000A0E610000001000000050000007EEDF5274BD1204...
6	0	1	0	0	trees	01030000A0E610000001000000050000FB970E7D7BD1204...
7	0	0	1	0	grassland	01030000A0E610000001000000050000000389527D2ABD1204...
8	0	0	1	0	grassland	01030000A0E610000001000000050000002AE54027DCD1204...
9	0	0	1	0	grassland	01030000A0E610000001000000050000000B9875A7C0CD2204...
10	0.0005	0.1597	0.8396	0	unknown	01030000A0E610000001000000050000000E77C74D13CD2204...
11	0	1	0	0	trees	01030000A0E610000001000000050000009CC48E266DD2204...
12	0	0	1	0	grassland	01030000A0E610000001000000050000000D35EA97B9DD2204...

Figure 27 - Capture d'écran de la base de données, extrait de la vue prediction_view.

La vue est générée à chaque nouveau clic sur un modèle.

La symbologie de la couche qui est affichée sur la carte est basée sur la colonne de la classe la plus probable (*most_likely_class*). Si la classe est indéterminée, « unknown », elle prend une couleur blanche avec une certaine transparence.

En passant le curseur de la souris sur la couche de classification, un graphique statistique en barres dans le volet de gauche s'affiche. Le score d'appartenance à chaque classe est affiché avec les mêmes couleurs que les classes représentées sur la carte. La solution implémentée utilise l'extension JavaScript Chart JS qui permet de concevoir des graphiques statistiques pour des pages Web (Chart.js, 2018).

4.3 Exemples de résultats de classification

Dans cet essai, l'accent a surtout été mis dans le développement de l'application Web plus que dans les résultats de la classification. L'amélioration et le perfectionnement de l'architecture d'un modèle est un travail de recherche en soi. Les images classifiées ont principalement été utilisées afin de tester l'application.

Ainsi, la précision de la classification importe moins et n'a pas été mesurée de manière rigoureuse en produisant, par exemple, une matrice de confusion avec des données de vérité terrain. Une évaluation visuelle est simplement effectuée en comparant la classification avec l'image classifiée. Les résultats permettent cependant d'apprécier le potentiel et certaines contraintes de l'apprentissage profond.

4.3.1 Merlischachen

L'image de Merlischachen, ne possédant que trois bandes (RGB), a été classifiée uniquement avec les modèles à trois bandes de DeepSat 4 et DeepSat 6. Ainsi, lorsque l'image est sélectionnée dans la section *Couches* du panneau de gauche, il est possible d'afficher les classifications suivantes :

- deepsat4-3bands
- deepsat6-3bands

Suite à la classification, l'image été découpée en 182 tuiles de 28 par 28 pixels (figure 28).

Avec le modèle basé sur DeepSat 4, la majorité des tuiles sont dans la classe « Herbe ». En comparant avec l'image de Merlischachen, il semble, en effet, que ces tuiles sont pour la plupart classifiées correctement.

La deuxième classe la plus présente est la classe « Autres ». Celle-ci regroupe tout ce qui n'est ni herbe, ni forêt, ni sol nu et correspond donc aux tuiles bâtiments, routes, eau, etc. La majorité de la zone urbaine de l'image ainsi que les routes se trouvent en effet dans la classe « Autres ».

En revanche, certaines tuiles de la zone urbaine ont été classifiées dans la classe « Forêt ». Il y a en effet quelques arbres dans la zone urbaine mais pas suffisamment pour justifier cette classe en forêt. Néanmoins, le peu de forêt présent sur l'image a été correctement classifié en forêt. Il s'agit de quelques tuiles dans le coin supérieur gauche de l'image, dans la première ligne.

Finalement, la classe « Sol nu » est la moins présente sur l'image. Les quelques tuiles identifiées sont en effet du sol nu, en revanche il en manque quelques-unes qui sont soit dans la classe « Herbe » soit indéterminées.

Concernant la classification avec DeepSat 6, la classe la plus présente est la classe « Forêt ». À nouveau, les quelques tuiles de forêt dans le coin supérieur gauche ont été classifiées correctement mais presque toute la zone urbaine a été classifiée faussement en forêt.

Figure 28 - Classification pour l'image de Merlischachen avec DeepSat 4 et DeepSat 6.

La deuxième classe la plus présente est la classe « Herbe ». La plupart des tuiles classifiées sont correctes, mais, il y quelques tuiles d'herbe qui sont indéterminées.

La troisième classe la plus présente est la classe « Sol nu ». À nouveau, la petite portion de sol nu dans le coin inférieur droit a été identifiée correctement et elle est un peu plus grande qu'avec DeepSat 4, ce qui est juste. En revanche, d'autres zones ont été trouvées alors que ce ne sont pas du sol nu.

La quatrième classe la plus présente est la classe « Routes ». Si quelques tuiles ont été identifiées correctement, la route principale qui traverse la ville ne l'a pas été. Aussi quelques tuiles urbaines ont été faussement identifiées en route.

Finalement, qu'une seule tuile « Bâtiment » a été trouvée alors que c'est une route, en revanche, aucune tuile « Eau » n'est présente, ce qui est juste.

On remarque aussi qu'il y a un peu plus de tuiles indéterminées avec le modèle DeepSat 4 qu'avec le modèle DeepSat 6.

4.3.2 Sherbrooke

L'image de Sherbrooke possédant quatre bandes spectrales, les modèles à 4 bandes de DeepSat 4 et DeepSat 6 ont été appliqués. Ainsi, lorsque l'image est sélectionnée dans la section *Couches* du panneau de gauche, il est possible d'afficher les classifications suivantes :

- deepsat4-4bands
- deepsat6-4bands

L'image a été découpée en 22 820 tuiles de taille 28 par 28 pixels (figure 29).

Le nombre de tuiles étant beaucoup plus important pour cette image que pour Merlischachen, il est impossible de présenter les résultats dans le détail, ainsi, les résultats principaux seront présentés ci-dessous.

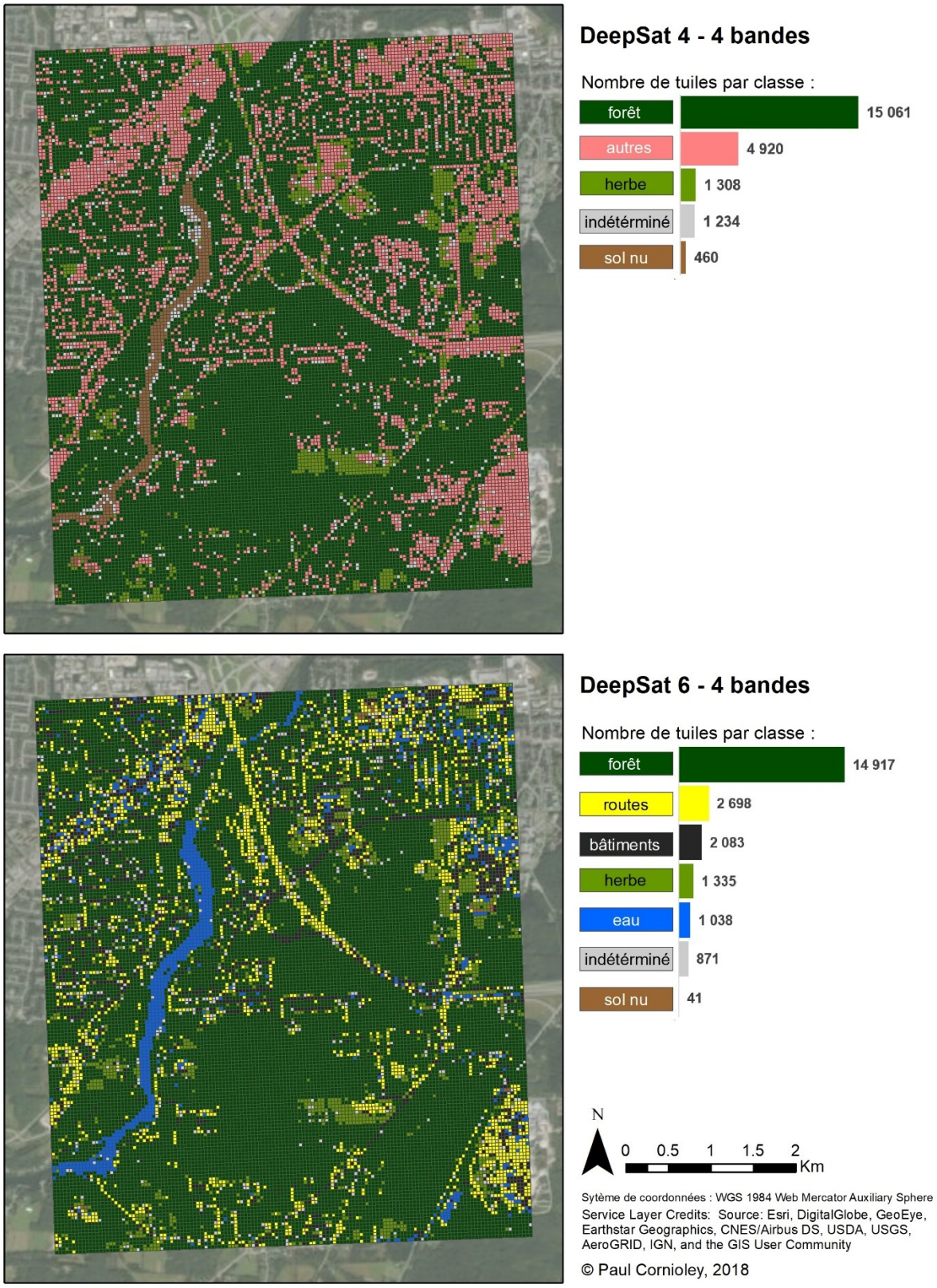


Figure 29 - Classification de l'image de Sherbrooke avec DeepSat 4 et DeepSat 6.

Avec les deux modèles, la classe la plus présente est la classe « Forêt » et cela correspond en effet à la réalité. En parcourant la carte, on observe que la plupart des tuiles forêt ont été classifiées correctement. Cependant, certains bâtiments de la zone résidentielle dans le nord-est de l'image ont été classifiés en forêt. Il y a beaucoup d'arbres dans la zone urbaine mais pas toujours suffisamment pour classifier celle-ci en forêt (figure 30).

Figure 30 - Capture d'écran d'une zone urbaine classifiée en forêt avec DeepSat 4.

La classe « Herbe » est également assez semblable entre les deux modèles. On reconnait sur les images à peu près les même grandes zones qui ont été classifiées pour la plupart correctement en herbe et le nombre de tuiles est assez proche : 1308 et 1335 pour DeepSat 4 et DeepSat 6 respectivement.

Avec DeepSat 4, la deuxième classe la plus présente est la classe « Autres », elle regroupe les routes, les bâtiments, l'eau, etc. On reconnait sur la classification les routes principales comme l'autoroute Jacques-O'Bready ou la route 216 mais aussi quelques routes secondaires. La zone résidentielle du nord-est et de l'arrondissement de Rock Forest–Saint-Élie–Deauville apparait également dans cette classe. Néanmoins, comme mentionné plus haut, beaucoup de bâtiments ont été classifié en forêt. Ainsi, un certain nombre de tuiles urbaines ne sont pas dans la bonne classe et il manque certains tronçons de routes.

Toujours avec DeepSat 4, on remarque que la rivière Magog est principalement classifiée en « Sol nu » et aussi partiellement indéterminée. En inspectant les tuiles indéterminées, elles possèdent pour la plupart un score assez élevé pour les classes « Forêt », « Herbe » et « Sol nu ».

Avec DeepSat 6, la rivière a été correctement classifiée en « Eau » dans sa partie en amont du barrage de Drummond. En revanche, pour la partie en aval, elle a été classifiée en un mélange des classes « Routes », « Forêt », « Herbe » ou indéterminée. Les tuiles indéterminées hésitent entre ces trois différentes classes.

On remarque aussi quelques tuiles « Eau » isolées et dispersées un peu partout sur l'image. Ces tuiles ne sont donc pas classifiées correctement puisqu'elles sont principalement des zones urbaines ou des routes.

Avec DeepSat 6, on retrouve la plupart des tuiles « Autres » de DeepSat 4 classifiées soit en « Bâtiments » soit en « Routes ». En revanche, le modèle de DeepSat 6 semble avoir du mal à différencier ces deux classes. En effet, les routes sont parfois classifiées en « Bâtiments » et le bâti en « Routes » (figure 31).

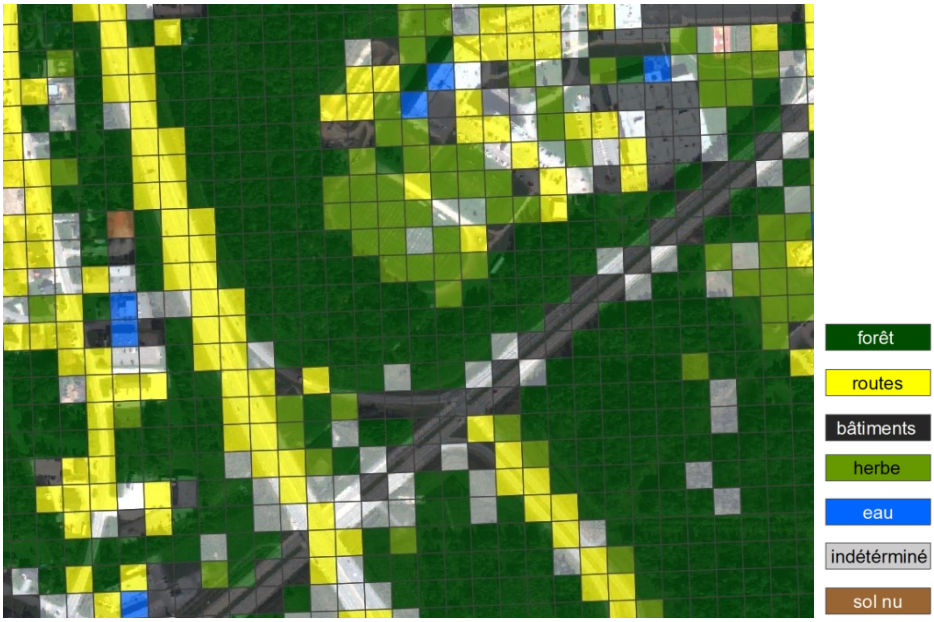


Figure 31 - Capture d'écran des classes « Bâtiments » et « Routes » interchangées.

Finalement, la classe « Sol nu » est la moins présente avec le modèle DeepSat 6. Seuls quelques toits de bâtiments de couleur orange ont été incorrectement classifiés en « Sol nu » (figure 32).

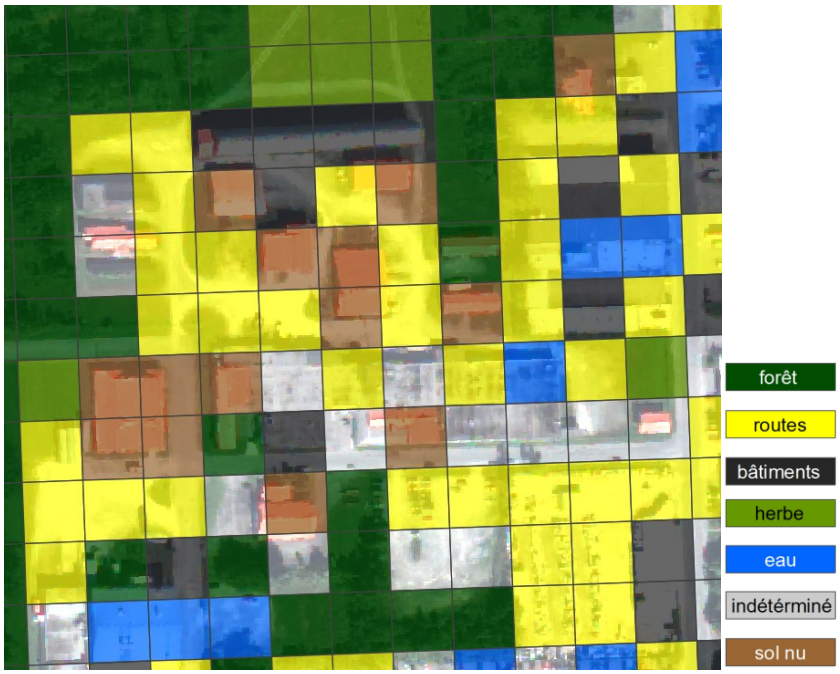


Figure 32 - Capture d'écran de toits orange de bâtiments classifiés en sol nu.

5. Discussion

5.1 Évaluation des modèles

À la figure 21, on constate que les pertes des données d'entrainement et de test diminuent conjointement. À noter qu'il y a quelques variations des pertes avec le modèle DeepSat 6 avec 4 bandes, mais elles ont des valeurs très faibles. Ceci signifie qu'il n'y a pas de surapprentissage, ce qui est confirmé avec les données de test, puisque la précision est toujours au-dessus de 99% (tableau 10). Ainsi, en appliquant presque le même modèle aux jeux de données on obtient une précision très élevée.

Cependant, il serait possible d'améliorer davantage ces modèles en les optimisant individuellement. La démarche serait d'améliorer graduellement le modèle en modifiant les couches et les paramètres jusqu'à obtenir du surapprentissage sur les données de validation. Lorsqu'il devient impossible d'affiner davantage le modèle, même en utilisant des couches de régularisation, ceci signifie que la capacité du modèle est atteinte. Ensuite, comme le surapprentissage n'est pas désirable, il faudrait revenir à une version précédente du modèle.

5.2 Évaluation de l'application

L'interface de l'application a été pensée pour être intuitive et conviviale. Elle est assez conventionnelle et s'inspire d'autres applications cartographiques en ligne telles que Google Maps ou OpenStreetMap, qui laissent toutes deux une grande place à la carte avec un panneau latéral sur la gauche. L'application se veut également très accessible puisqu'elle permet de classifier une image géoréférencée sans connaissance préalable en apprentissage profond.

L'exploration des résultats de la classification est également intuitive avec la symbologie de la classe la plus probable. Il est aussi possible de parcourir en détail le score de chaque tuile en passant la souris dessus. Cela permet notamment d'observer les tuiles qui n'ont pas de classe déterminée pour voir entre quelles classes la classification a hésité.

Il serait possible d'améliorer l'accessibilité de l'application avec, par exemple, des boutons d'aide et d'information. Ceux-ci permettraient de donner plus de précisions sur chaque section du panneau principal. Par exemple, avec des conseils sur comment choisir le jeu de données le plus adapté.

Du côté des technologies, uniquement des solutions libres et ouvertes ont été utilisées, respectant ainsi la contrainte imposée pour assurer une pérennité du système et une réutilisation simplifiée

pour la communauté. De plus, le principe d'interopérabilité a également été appliqué puisque les données du serveur cartographique sont disponibles dans les services WMS (Web Map Service) et WFS (Web Feature Service). Par exemple, il est possible d'ajouter la couche contenant les prédictions en WFS dans QGIS (figure 33).

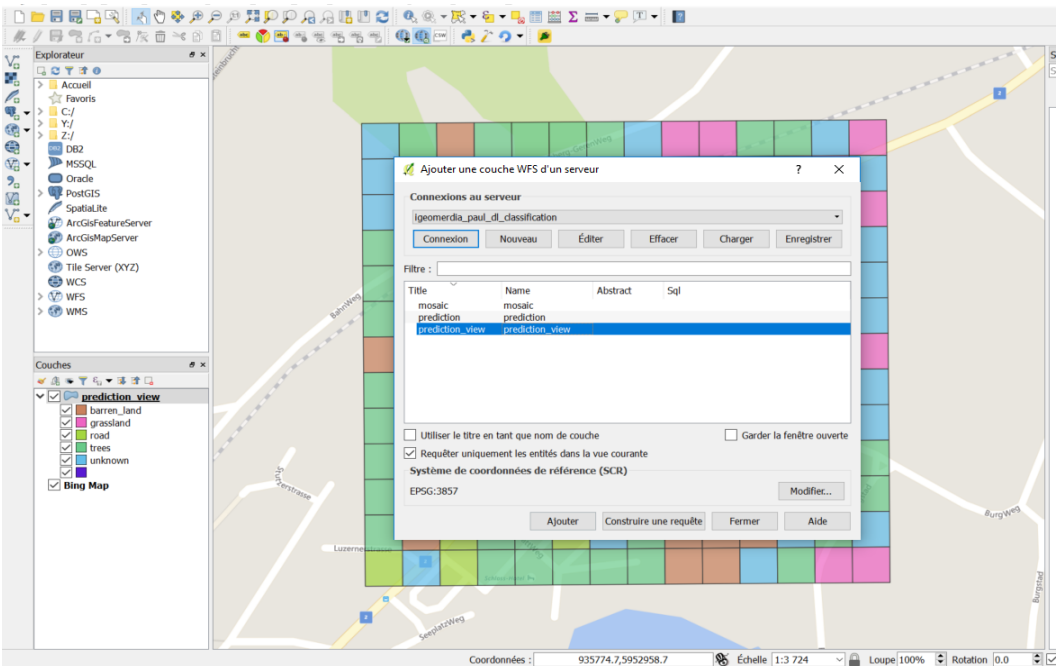


Figure 33 - Ajout de la couche contenant la classification en WFS dans QGIS.

Le modèle conceptuel proposé dans la méthodologie expose les éléments d'un projet d'apprentissage profond pour classifier des images. Ces étapes seront très semblables pour n'importe quel autre projet. Il faut posséder ou constituer un jeu de données, séparer ces données en trois (données d'entrainement, de validation et de test), créer un modèle, l'entrainer et le tester pour ensuite faire des prédictions sur les images à classifier. Concernant le stockage, le modèle physique des données est personnalisé pour ce projet, c'est-à-dire, pour classifier des images géoréférencées qui ont été découpées en tuiles. Néanmoins, ce modèle pourrait être répété dans un projet similaire mais avec des images provenant d'autres jeux de données comportant d'autres classes. Par ailleurs, le stockage n'est pas dépendant de la librairie d'apprentissage. Il serait possible d'entrainer un modèle avec une autre librairie, il faudrait néanmoins prêter attention au format utilisé pour les tenseurs.

Par simplicité, ce projet utilise le format de matrice de type NumPy mais il serait envisageable d'utiliser un autre format, plus efficient si les données sont souvent sollicitées. Pour l'instant, dans l'application, ces données sont seulement chargées lors de l'exploration des jeux de données, lorsque l'on clique sur une classe pour afficher des extraits d'images d'une classe. Si ces données sont manipulées plus souvent comme pour l'ajout de nouvelles images, il faudrait peut-être utiliser un autre format plus efficace.

5.3 Évaluation des résultats de la classification

Tout d'abord, on remarque que les tuiles appartenant à certaines classes sont généralement classifiées plus justement que d'autres. C'est le cas des tuiles des classes naturelles (« Forêt », « Herbe », « Sol nu » et « Eau ») qui sont souvent classifiées correctement. Ceci s'explique principalement pour deux raisons.

Premièrement, la plupart de ces tuiles sont constituées à 100% d'une seule classe. Les classes naturelles étant plus étendues et homogènes, elles sont plus souvent découpées en tuiles composées d'une seule classe. À l'inverse, les zones urbaines sont des zones résidentielles peu denses qui sont un mélange de bâti, de routes, d'arbres, d'herbe, etc. Concernant les routes, elles sont souvent trop étroites pour représenter une classe unique sur les tuiles de 28 par 28 mètres.

Comme les images des jeux de données sont composées d'une seule classe, le modèle est entrainé pour reconnaître qu'une seule classe par tuile, et donc, le modèle performe mieux pour classifier des tuiles possédant une seule classe. Par ailleurs, on remarque que lorsqu'une tuile de classe anthropique (« Bâtiment » ou « Routes ») est constituée d'une seule classe, elle est souvent classifiée correctement.

Deuxièmement, les jeux de données de cet essai ne proviennent pas de la même région géographique que les deux images classifiées. En effet, les images des jeux de données sont constituées d'images aériennes de la Californie (États-Unis) alors que les images classifiées sont de la ville de Sherbrooke (Québec) et d'un petit village en Suisse. Les classes naturelles des images classifiées sont plus similaires à celles des jeux de données que les classes anthropiques. Par exemple, les forêts de la Californie sont visuellement assez proches de celles de Sherbrooke et aussi de la Suisse (figure 34). Alors que la classe « Bâtiments » est assez différente de Sherbrooke et encore plus de l'image de Suisse. De façon similaire, les routes des jeux de données (classe « Routes » pour DeepSat 6 et certaines images de la classe « Autres » de DeepSat 4) sont souvent

de grands boulevards qui sont peu présents sur l'image de Sherbrooke et pas du tout sur l'image de Suisse.

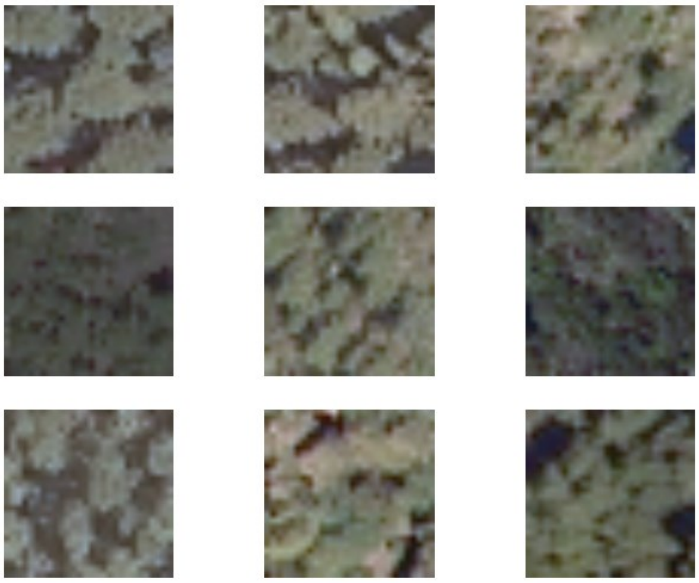


Figure 34 - Extraits de la classe « Forêt » de DeepSat 6.

Cette différence entre les images d'entrainement (de la base de données) et celle de test (à classifier) a notamment été relevée par Dahmane *et al.* (2017) qui ont observé une précision un peu meilleure sur des images de test de Vancouver que sur celles de la ville de Québec avec des modèles entrainés uniquement sur des images de Vancouver.

Ainsi, pour de meilleurs résultats, il faut que les images d'entrainement soient visuellement proches de l'image à classifier. C'est pour cela qu'une partie de l'interface de l'application est dédiée à l'exploration des jeux de données disponibles sur le serveur, cela permet de sélectionner le jeu de données le plus approprié pour classifier l'image. Néanmoins, il y a une exception aux classes naturelles : l'eau. Celle-ci a une apparence texturée étrange dans les images des deux bases de données et semble contenir beaucoup de bruit (figure 35). Elle ressemble assez peu à l'eau présente sur l'image de Sherbrooke, principalement représentée par la rivière Magog. La partie en amont du barrage de la rivière Magog a cependant été classifiée correctement avec DeepSat 6. Avec DeepSat 4, l'eau devrait se trouver dans la classe « Autres », mais la rivière est un mélange de classe « Sol nu » et indéterminée.

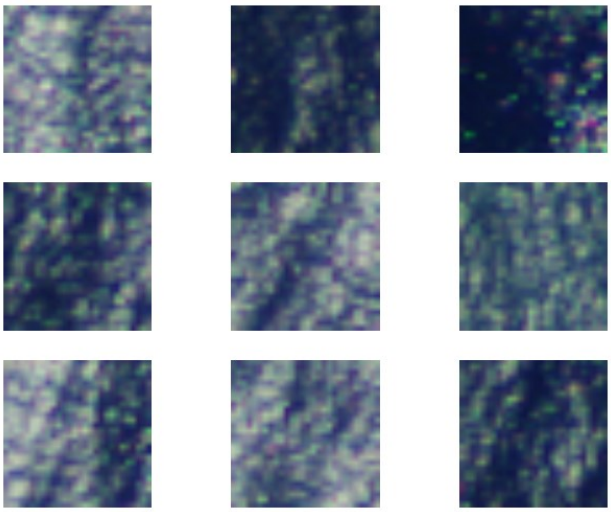


Figure 35 - Extrait de la classe « Eau » de DeepSat 6.

De plus, avec DeepSat 6, il y a des tuiles « Eau » qui sont dispersées à différents endroits de l'image de Sherbrooke. Ce sont souvent des zones urbaines ou des routes, qui ont également une faible réflectance dans le proche infrarouge. Cette bande n'a donc pas pu aider à différencier ces zones.

À noter que ce problème ne se remarque pas avec DeepSat 4, puisque l'eau est dans la classe « Autres » avec les bâtiments et les routes. Il n'est donc impossible de savoir si le modèle a réussi à distinguer l'eau des bâtiments ou des routes.

Ensuite, on remarque des différences entre les classifications basées sur DeepSat 4 et DeepSat 6. Pour ce qui est de Sherbrooke, les classes « Forêt » et « Herbe » sont assez semblables avec les deux modèles. Cela fait du sens car, bien que les images proviennent de jeux de données différents, les images des classes correspondantes sont visuellement assez proches entre ces jeux de données puisqu'elles proviennent de la même région géographique.

En revanche, pour la classe « Eau », la majorité du tracé de la rivière Magog est classifiée correctement avec DeepSat 6 alors qu'avec le modèle de DeepSat 4, la rivière est un mélange de « Sol nu » et indéterminée. Il est difficile de savoir si des images d'eau sont bien présentes dans la classe « Autres » de DeepSat 4 (figure 36). Probablement que ces exemples sont trop différents de l'eau de l'image pour la classifier correctement. En revanche, les bassins d'eau proches de la carrière Bel Horizon ont été classifiés correctement en « Autres » (figure 37). Il est à nouveau impossible de savoir si le modèle a effectivement reconnu de l'eau ou autre chose.

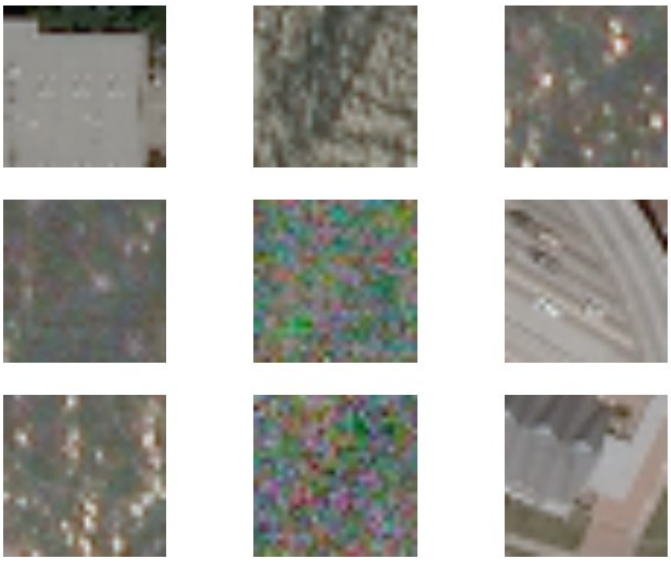


Figure 36 - Extraits de la classe « Autres » de DeepSat 4.

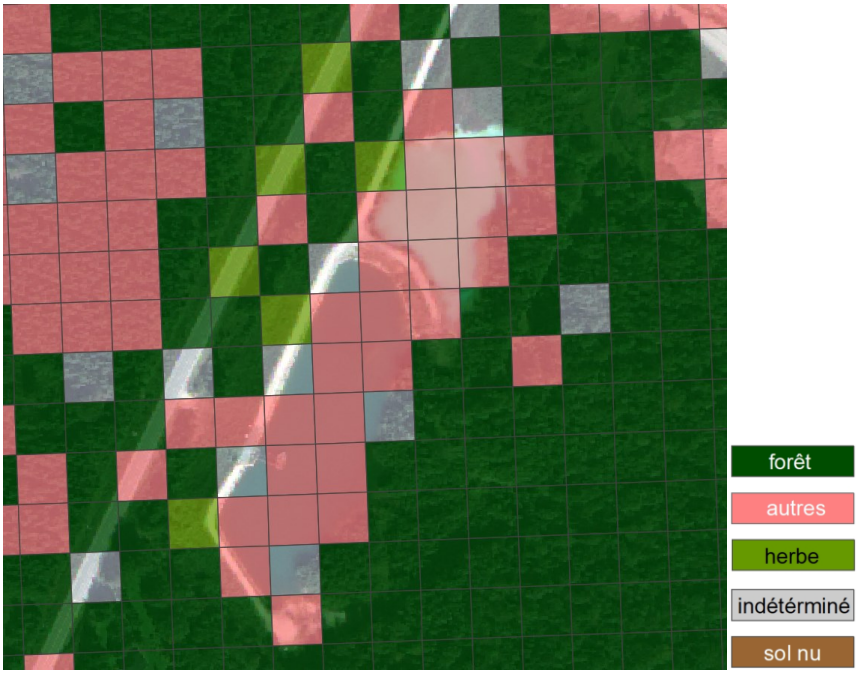


Figure 37 - Capture d'écran de bassins d'eau à proximité de la carrière Bel Horizon. Classification de l'image de Sherbrooke avec DeepSat 4.

Néanmoins, pour le reste de la classe « Autres », il semble que les résultats sont globalement meilleurs avec DeepSat 4. Cette classe regroupe tout ce qui n'est ni forêt, sol nu ou herbe. Avec moins de différenciation, il est peut-être moins facile de se tromper. Par exemple, avec DeepSat 6, on remarque que les routes sont parfois classifiées en urbain et vice-versa.

Au niveau de la qualité des jeux de données, il semble que des images de bâtiments peuvent porter à confusion puisque des bâtiments ressemblent à des routes, voir l'image de droite de la figure 38 les tuiles en haut à droite et en bas à droite.

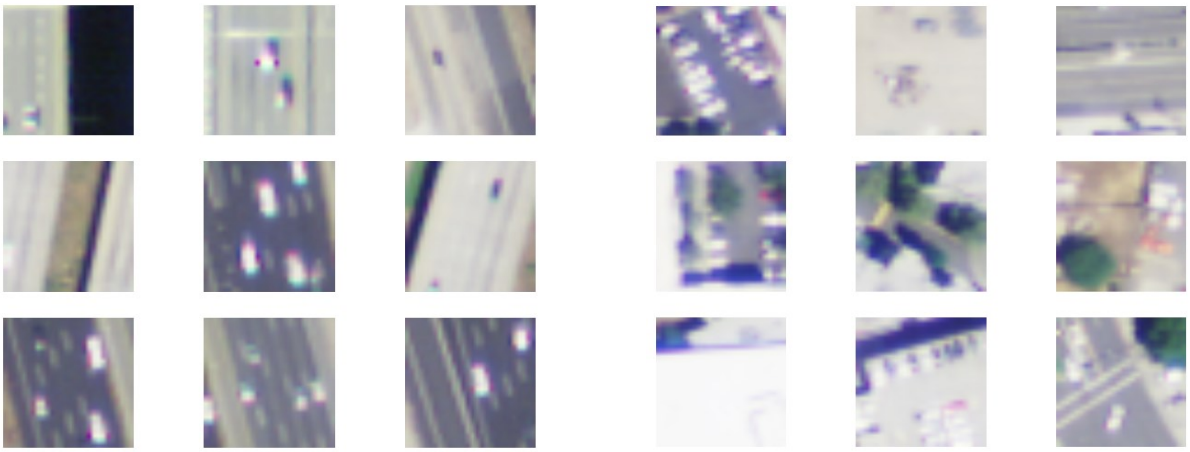


Figure 38 - Extraits du jeu de données DeepSat 6, à gauche la classe « Routes » et à droite la classe « Bâtiments ».

L'absence de classe « Eau » dans DeepSat 4 permet aussi d'éviter des tuiles eau isolées un peu partout dans l'image puisque les tuiles sont associées dans la classe « Autres ». Ainsi, moins de classe donne des résultats plus justes mais il y a moins de distinction entre les classes.

Concernant l'image de Suisse, les résultats sont plus cohérents avec DeepSat 4 qu'avec DeepSat 6. Avec DeepSat 6, la zone urbaine de l'image de Suisse a été presque systématiquement classifiée en forêt. Il y a en effet quelques arbres dans le village mais cela n'explique pas l'imprécision pour toutes les tuiles. Il faut rappeler que ce modèle n'utilise pas la bande proche infrarouge alors que ceci aurait pu aider à discriminer le bâti de la végétation puisque leur réflectance dans cette longueur d'onde est très différente. Néanmoins, avec le modèle à 3 bandes basé sur DeepSat 4, la zone urbaine a été correctement classifiée dans la classe « Autres ». Il semble donc possible

d'obtenir de bons résultats même avec seulement 3 bandes spectrales. Une investigation plus approfondie serait nécessaire pour mesurer l'apport de la bande proche infrarouge. Il faudrait par exemple comparer la classification d'une même image qui possède une bande proche infrarouge et la classifier avec un modèle qui utilise cette bande et un modèle qui ne l'utilise pas. L'image de Sherbrooke pourrait être utilisée avec une vérité terrain pour obtenir des statistiques et une matrice de confusion.

Finalement, on observe que lorsque qu'une tuile est assignée à une classe son score d'appartenance à cette classe est souvent très élevé : proche ou égal à 100%. Ceci est correct lorsque la classe est juste, en revanche, il arrive que le score soit très élevé avec une mauvaise classe. Ainsi, il n'est pas possible de faire totalement confiance au score élevé d'une tuile puisque la classe peut être fausse. Donc, bien que les modèles possèdent des précisions très hautes avec les images de test, plus de 99%, cela ne veut pas dire que la précision sera aussi bonne avec des images d'un environnement différent.

5.4 Limites du système et recommandations

Dans son état actuelle, l'application n'est pas totalement fonctionnelle puisqu'il manque la possibilité d'ajouter des images géoréférencées sur le serveur. Comme mentionné dans la section des résultats, le temps a manqué pour développer cette fonctionnalité. D'autre part, la classification par apprentissage profond d'une nouvelle image n'est pas possible puisque Keras n'est pas installé sur le serveur. Il faudrait migrer l'application sur un serveur qui possède Keras avant d'implémenter cette fonctionnalité. De plus, il faudrait ajouter la possibilité le télécharger les couches classifiées directement dans l'application. Présentement, il est possible de télécharger ces couches mais il faut d'abord générer la vue dans l'application puis la télécharger avec le service WFS, avec QGIS par exemple. À nouveau le temps a manqué pour développer cette fonctionnalité.

Télécharger les couches classifiées, permettrait de poursuivre l'amélioration de la classification. Par exemple, la classification par apprentissage profond ne tient pas compte du voisinage des tuiles. Il est plus probable que des tuiles de même classe soient regroupées plutôt que dispersées, par exemple, une tuile route isolée au milieu d'une forêt est peu probable. Ainsi, en utilisant les scores d'appartenance calculés à l'aide de l'apprentissage profond, il serait possible de résoudre les tuiles de classe indéterminée en utilisant le voisinage.

Une des contraintes de l'application est que les images doivent respecter certains critères afin d'être classifiées. Ces critères dépendent surtout des modèles disponibles sur le serveur. Notamment, la résolution spatiale doit être au maximum la même que celle des images du jeu de données, pour DeepSat 4 et 6, elle est de 1 mètre. Il est possible de classifier des images avec une résolution spatiale plus petite (par exemple, 5 cm pour Merlischachen) mais elles doivent d'abord être rééchantillonnées à la résolution spatiale des images du jeu de données. À noter qu'il est techniquement possible de classifier une image avec une résolution spatiale plus grande que celles des jeux de données, mais les résultats seraient moins bons. Ainsi, une fonctionnalité qui pourrait être ajoutée à l'application est la possibilité de rééchantillonner les images à la volée lorsqu'un jeu de données dont la résolution spatiale des images est plus grande que celle de l'image à classifier. Ceci permettait de classifier les images avec davantage de modèles sans devoir traiter ces images avant de les télécharger sur l'application.

Une autre limitation de l'application est qu'il n'est pas possible de paramétrer la classification. Si le résultat de la classification n'est pas satisfaisant, il est seulement possible de sélectionner un autre modèle pour classifier l'image pour essayer d'obtenir de meilleurs résultats. Comme il a été vu dans les résultats, la précision dépend beaucoup de la ressemblance des images du jeu de données avec l'image à classifier. En conséquence, on pourrait imaginer la possibilité d'ajouter des tuiles de l'image classifiée au jeu de données pour étoffer celui-ci. Ainsi, si après la classification, les résultats ne sont pas satisfaisants, il serait possible de sélectionner des tuiles sur la carte pour les ajouter au jeu de données. Pour améliorer les classes qui performent moins bien, il faudrait en particulier ajouter des exemples de tuiles qui n'ont pas été classifiées correctement.

Par ailleurs, de manière intuitive, posséder une grande variation des images dans les classes permettrait de classifier avec plus de précision des images variées et d'environnements différents. C'est également le constat qui a été fait par Turgeon-Pelchat (2018) : en ajoutant des images de l'environnement à classifier au jeu de données la classification est grandement améliorée. Ainsi, il serait plus pertinent de posséder un jeu de données de taille moyenne avec des images variées pour les classes plutôt qu'un jeu de données importants mais avec des images très similaires.

Comme le jeu de données contient généralement déjà un grand nombre d'images, pour que les tuiles nouvellement ajoutées ne soient pas sous-représentées dans la classe, il est possible d'augmenter artificiellement le nombre d'exemples. Il serait possible d'implémenter une stratégie

qui est souvent utilisée lorsque les jeux de données à entrainer sont trop petits : la technique du *data augmentation*. Cela consiste à appliquer des transformations à une image pour en générer de nouvelles qui sont fortement corrélées, par exemple : des rotations, étirements, zoom, etc. (figure 39). Cette fonctionnalité permettrait de démultiplier les images ajoutées sans devoir ajouter manuellement beaucoup d'exemples.

Figure 39 - Technique de data augmentation sur des images, extraite de Chollet (2018).

Une fois que les tuiles sont ajoutées au jeu de données, il est nécessaire d'entrainer à nouveau le modèle pour prendre en compte les nouvelles images. Cette opération demande beaucoup de capacité de traitement selon la taille du modèle et du jeu de données.

Concernant la performance, il y a quelques points à améliorer. Par exemple, l'affichage des images sur la carte Web, en particulier celle de Sherbrooke, prend quelques secondes. Les images sont affichées avec leur résolution maximale peu importe le niveau de zoom, il faudrait créer des tuiles avec différentes résolutions pour accélérer l'affichage.

Si l'application est destinée à évoluer vers de nouvelles technologies, il serait probablement préférable de l'adapter à un cadre de développement de GeoDjango. C'est le cadre de développement de référence pour travailler avec des scripts en Python sur un serveur.

Aussi, dans le cas où l'application peut avoir plusieurs utilisateurs en même temps, il faudrait s'assurer qu'il y n'ait pas de conflits. Par exemple, lors de la création des images d'exemples des classes, si deux utilisateurs cliquent l'un après l'autre sur une classe mais que l'image n'a pas fini d'être créée, l'image ne sera pas chargée correctement.

6. Conclusion

Dans ce travail, un module d'apprentissage profond a été intégré à un SIG sur le Web. En particulier, une application permettant de classifier des images géoréférencées à l'aide de réseaux de neurones convolutifs a été développée. La solution repose uniquement sur des technologies de programmation libres et ouvertes pour assurer la pérennité du système et permettre sa réutilisation par la communauté. Par ailleurs, les normes internationales pour la diffusion de l'information ont été respectées s'assurant ainsi l'interopérabilité du système.

L'application a été conçue pour être conviviale et ne nécessitant pas de connaissance en apprentissage profond pour exécuter la classification d'image. L'interface Web est axée autour de trois fonctionnalités :

- l'ajout d'une image à classifier;
- l'exploration des jeux de données et des modèles d'apprentissage profond disponibles pour ensuite classifier une image par apprentissage profond;
- et l'exploration des résultats de la classification.

Un modèle conceptuel a également été élaboré et expose les étapes permettant de classifier une image par apprentissage profond. Il montre notamment les données nécessaires et comment elles doivent être divisées pour entrainer un modèle.

Plusieurs librairies d'apprentissage profond dans le langage de programmation Python ont été comparées afin de sélectionner celle a été utilisée dans cet essai. Seules des solutions libres et ouvertes ont été retenues. Le choix s'est porté sur la librairie Keras, notamment pour sa facilité de prise en main.

Cette librairie a permis de développer des modèles d'apprentissage profond et de les entrainer avec deux jeux de données possédant 500 000 et 405 000 images. Pour chaque jeu de données, deux modèles ont été entrainés : l'un avec les bandes RGB plus la bande du proche infrarouge et l'autre avec seulement les bandes RGB. Une précision au-dessus de 99% a été obtenue avec chaque modèle sur les données de test.

Par la suite, les modèles entrainés ont servi à classifier deux images géoréférencées : une qui possède la bande proche infrarouge et une autre ne possédant que les bandes RGB. Les classes naturelles comme la « Forêt » et l'« Herbe » ont montré les meilleurs résultats surtout lorsque la

bande proche infrarouge est utilisée. En revanche, les classes anthropiques telles que les « Routes » et les « Bâtiments » ont, de manière générale, montré une précision plus faible. Ces résultats s'expliquent en partie par des différences visuelles importantes selon les classes entre les images classifiées et celles des jeux de données. Ainsi, un des enjeux pour obtenir de bons résultats par apprentissage profond est de posséder un jeu de données adapté à l'image à classifier, c'est-à-dire, il faut qu'il existe dans le jeu de données des exemples d'images qui sont proches de l'image à classifier.

Dans une version future de l'application, le développement d'une fonctionnalité permettant l'ajout d'images classifiées permettrait d'augmenter considérablement la performance de la classification en enrichissant les jeux de données avec des exemples variés.

7. Références

Adivarekar B. (2018) simple Keras CNN with 95.13% accuracy. https://www.kaggle.com/bhumitadivarekar/simple-keras-cnn-with-95-13-accuracy

Amazon.com Inc. (2018) Amazon Alexa. https://developer.amazon.com/fr/alexa

BAIR (2018) Caffe | Deep learning Framework. https://caffe.berkeleyvision.org/

Basu, S., Ganguly, S., Mukhopadhyay, S., DiBiano, R., Karki, M. et Nemani, R. (2015) Deepsat: a learning framework for satellite imagery. ACM SIGSPATIAL 2015, 22 p.

Basu, S., Ganguly, S., Mukhopadhyay, S., DiBiano, R., Karki, M. et Nemani, R. (2018) SAT-4 and SAT-6 airborne datasets. http://csc.lsu.edu/~saikat/deepsat/

Belspo (2018) EOEdu - Petit guide de la télédétection - Classification non supervisée http://eoedu.belspo.be/fr/guide/clasnons.asp?section=3.6.1

Bloomberg (2018) Google Turning Its Lucrative Web Search Over to AI Machines. https://www.bloomberg.com/news/articles/2015-10-26/google-turning-its-lucrative-web-search-over-to-ai-machines

Bouroubi Y., P. Bugnet, T. Nguyen-Xuan, C. Gosselin et M. Benoit (2017) Rooftop characterization by applying deep learning to WorldView-3 images: application to technical solar photovoltaic assessment. Sommet OT 2017, 20 au 22 juin, Montréal, QC, Canada, 10 p.

Carleer, A. P., Debeir, O., et Wolff, E. (2005). Assessment of very high spatial resolution satellite image segmentations. Photogrammetric Engineering & Remote Sensing, vol. 71, nº 11, p. 1285-1294.

Chart.js (2018) Open source HTML5 Charts for your website. http://www.chartjs.org/

Chollet, F. (2018) Deep Learning with Python. Manning Publications Co., 350 p.

Cooper, S. B. et Van Leeuwen, J. (2013) Alan Turing: His work and impact. Elsevier, 944 p.

Dahmane M., Foucher S., Beaulieu M., Riendeau F., Bouroubi Y. et Benoit M., (2016) Object Detection in Pleiades Images using Deep Features. IGARSS 2016, 10 au 15 Juillet, Pékin, Chine, 4 p.

DataCamp (2018) Keras Tutorial: Deep Learning in Python (article) – DataCamp. https://www.datacamp.com/community/tutorials/deep-learning-python

DigitalGlobe (2015) WorldView-3, image satellite de Sherbrooke.

ESA (2018) WorldView-3 - Satellite Missions - eoPortal Directory. https://directory.eoportal.org/web/eoportal/satellite-missions/v-w-x-y-z/worldview-3

ESRI (2015) ArcGIS 10.3.1. http://desktop.arcgis.com/fr/arcmap/10.3/main/get-started/whats-new-in-arcgis-1031.htm

ESRI (2018) ArcGIS Geostatistical Analyst | Model Spatial Data & Uncertainty. https://www.esri.com/en-us/arcgis/products/geostatistical-analyst/overview

Foody, G. M., McCulloch, M. B., et Yates, W. B. (1995). Classification of Remotely Sensed Data by an Artificial Neural Network: Issues Related to Training Data. Photogrammetric Engineering & Remote Sensing, vol. 61, nº 4, p. 391-401.

Free Software Foundation, Inc. (2016) QGIS 2.18.13. https://docs.qgis.org/2.8/en/docs/

FSO (2018) Regionalprofile. http://www.media-stat.admin.ch/maps/profile/profile.html?226.1331.en.geoRefStandard

Google Brain (2018) TensorFlow. https://www.tensorflow.org/

Haklay, M., Singleton, A., et Parker, C. (2008). Web mapping 2.0: The neogeography of the GeoWeb. Geography Compass, vol. 2, nº 6, p. 2011-2039.

Harris (2018a) Documentation Center. https://www.harrisgeospatial.com/docs/NeuralNet.html

Harris (2018b) ENVI. https://www.harris.com/solution/envi

Hastie, T., Tibshirani, R. et Friedman, J. (2009) The Elements of Statistical Learning; Data Mining, Inference and Prediction. Springer, 757 p.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... et Kingsbury, B. (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, vol. 29, nº 6, p. 82-97.

Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R., Mujica, F., Coates, A. Y., Ng (2015) An Empirical Evaluation of Deep Learning on Highway Driving. 7 p.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... et Darrell, T. (2014) Caffe: Convolutional architecture for fast feature embedding. 22nd ACM international conference on Multimedia, p. 675-678).

Kaggle (2018) Kaggle: Your Home for Data Science. https://www.kaggle.com/

Krizhevsky, A., Sutskever, I., et Hinton, G. E. (2012) Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, p. 1097-1105.

Lambin P. (2017) MILA and the future of Theano. https://groups.google.com/forum/#!msg/theano-users/7Poq8BZutbY/rNCIfvAEAwAJ

LeCun, Y. (1985) Une procédure d'apprentissage pour réseau à seuil asymétrique. Cognitiva 85 : à la Frontière de l'Intelligence Artificielle, des Sciences de la Connaissance et des Neurosciences, p. 599-604.

LeCun, Y., Bengio, Y., et Hinton, G. (2015). Deep learning. Nature, vol. 521, nº 7553, p. 436-444.

Lisa Lab. (2018) Welcome – Theano 1.0.0 documentation. http://www.deeplearning.net/software/theano/

PCI Geomatics (2018) PCI Geomatics - PCI Geomatics. http://www.pcigeomatics.com/

Peterson, M. P. (2012). Online Maps with APIs and Web Services, Springer, 317 p.

Pix4D (2018) Pix4Dmapper: Photogrammetry software. Desktop or cloud processing. https://pix4d.com/product/pix4dmapper-photogrammetry-software/

PyImageSearch (2018) My Top 9 Favorite Python Deep Learning Libraries – PyImageSearch. https://www.pyimagesearch.com/2016/06/27/my-top-9-favorite-python-deep-learning-libraries/

PyTorch (2018) PyTorch. https://pytorch.org/

Rumelhart, D. E., Hinton, G. E., et Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, vol. 323, nº 6088, p. 533.

senseFly (2018a) Datasets – senseFly. https://www.sensefly.com/education/datasets/

senseFly (2018b) senseFly - The Professional's Mapping Drone of Choice. https://www.sensefly.com/

Skymind (2018) Deep Learning Comp Sheet: Deeplearning4j vs. Torch vs. Caffe vs. TensorFlow vs. MxNet vs. CNTK - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM. https://deeplearning4j.org/compare-dl4j-tensorflow-pytorch

Steiniger, S., Hunter, A. J. (2013) The 2012 free and open source GIS software map–A guide to facilitate research, development, and adoption. Computers, environment and urban systems, vol. 39, p. 136-150.

Taigman, Y., Yang, M., Ranzato, M. A., et Wolf, L. (2014) Deepface: Closing the gap to human-level performance in face verification. IEEE conference on computer vision and pattern recognition, p. 1701-1708.

Tang, J., Deng, C., Huang, G. B., & Zhao, B. (2015) Compressed-domain ship detection on spaceborne optical image using deep neural network and extreme learning machine. IEEE Transactions on Geoscience and Remote Sensing, vol. 53, nº 3, p. 1174-1185.

The MathWorks Inc. (2018) Find local minima - MATLAB islocalmin. https://www.mathworks.com/help/matlab/ref/islocalmin.html

Turgeon-Pelchat, M. (2018) Extraction d'informations cartographiques à partir d'imagerie satellitaire haut résolution. Conférence organisée par le CARTEL et le département de géomatique appliquée, 2018, 19 avril, Sherbrooke, QC, Canada.

Turing A. M. (1950) Computing Machinery and Intelligence. Mind vol. 49, p. 433-460.

Ville de Montréal (2016) Annuaire statistique de l'agglomération de Montréal – 2016. http://ville.montreal.qc.ca/

Ville de Sherbrooke (2018) Statistiques sur la ville – Ville de Sherbrooke. https://www.ville.sherbrooke.qc.ca/visiteur/statistiques-sur-la-ville/

Werbos, P. (1974) Beyond regression: new fools for prediction and analysis in the behavioral sciences. Thèse de doctorat, Université de Harvard, 453 p.

Zhang, L., Zhang, L., et Du, B. (2016) Deep learning for remote sensing data: A technical tutorial on the state of the art. IEEE Geoscience and Remote Sensing Magazine, vol. 4, nº 2, p. 22-40.

8. Annexes

Annexe 1 - Script Python entrainement du modèle DeepSat 6 avec 4 bandes

In [1]:
```
#Importation des libraries

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # pour faire des graphes

from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout # components of network
from keras import models, layers

# not needed in Kaggle, but required in Jupyter
%matplotlib inline
```

```
Using TensorFlow backend.
```

1. Préparation des données

In [3]:
```
# Importation des images d'entrainement
train_x = np.load('../INPUT_DATA/FilesForDataBase/deepsat6/train_x_sat6.npy')

#Importation des labels d'entrainement
train_y = np.load('../INPUT_DATA/FilesForDataBase/deepsat6/train_y_sat6.npy')

# Nombre de bits des images (255 = 8 bits)
nbBits = 255

# Découpage des images d'entrainement
# Tous les pixels < 0 seront mis à 0
# Tous les pixels > nbBits seront mis à nbBits
train_x = train_x.clip(0,nbBits)

# Rééchelonnage de la valeur des pixels
train_x = train_x.astype('float32') / nbBits

# Suppression de la bande NIR des images
#train_x = train_x[:,:,:,:3]


# Changement du type des labels
train_y = train_y.astype('float32')
```

In [9]:
```
# Importation des images de test
test_x = np.load('../INPUT_DATA/FilesForDataBase/deepsat6/test_x_sat6.npy')

#Importation des labels de test
test_y = np.load('../INPUT_DATA/FilesForDataBase/deepsat6/test_y_sat6.npy')

# Nombre de bits des images (255 = 8 bits)
nbBits = 255

# Découpage des images de test
# Tous les pixels < 0 seront mis à 0
# Tous les pixels > nbBits seront mis à nbBits
test_x = test_x.clip(0,nbBits)

# Rééchelonnage de la valeur des pixels
test_x = test_x.astype('float32') / nbBits

# Suppression de la bande NIR des images
#test_x = test_x[:,:,:,:3]


# Changement du type des labels
test_y = test_y.astype('float32')
```

1.1 Séparation des données d'entrainement

In [3]:
```
# --- Pas nécessaire ici, voir les paramètres de l'entrainement du modèle ----

# Séparation des données d'entrainement

# (partial_x_train, partial_y_train), (x_val, y_val) = splitTrainingData(x_train_4D, y_train)
```

2. Définition du modèle

In [14]:
```
height = 28
width = 28
nbChannels = 4

nbClasses = 6

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(height, width, nbChannels)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(nbClasses, activation='softmax'))
```

2.1 Compilation du modèle

In [15]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        1184
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390
=================================================================
Total params: 93,926
Trainable params: 93,926
Non-trainable params: 0
_________________________________________________________________
```

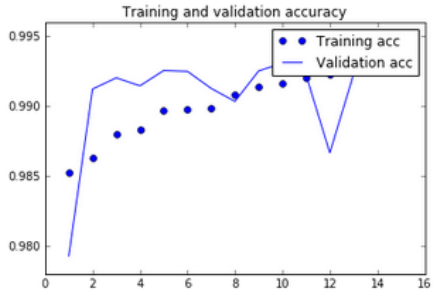
3. Entrainement du modèle

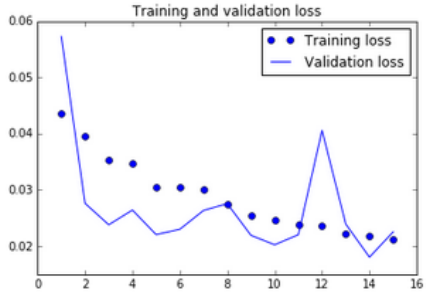
In [19]:
```
# validation_split permet de séparer les données d'entrainement

history = model.fit(train_x,
                    train_y,
                    epochs=15,
                    batch_size=128,
                    validation_split=0.2)
```

```
Train on 259200 samples, validate on 64800 samples
Epoch 1/15
259200/259200 [==============================] - 15s 59us/step - loss: 0.0435 - acc: 0.9852 - val_loss: 0.0573 -
val_acc: 0.9793
Epoch 2/15
259200/259200 [==============================] - 15s 59us/step - loss: 0.0396 - acc: 0.9863 - val_loss: 0.0276 -
val_acc: 0.9912
Epoch 3/15
259200/259200 [==============================] - 15s 60us/step - loss: 0.0353 - acc: 0.9880 - val_loss: 0.0238 -
val_acc: 0.9920
Epoch 4/15
259200/259200 [==============================] - 15s 59us/step - loss: 0.0348 - acc: 0.9883 - val_loss: 0.0264 -
val_acc: 0.9914
Epoch 5/15
259200/259200 [==============================] - 15s 60us/step - loss: 0.0305 - acc: 0.9897 - val_loss: 0.0221 -
val_acc: 0.9925
Epoch 6/15
259200/259200 [==============================] - 15s 60us/step - loss: 0.0306 - acc: 0.9898 - val_loss: 0.0230 -
val_acc: 0.9925
Epoch 7/15
259200/259200 [==============================] - 15s 60us/step - loss: 0.0300 - acc: 0.9899 - val_loss: 0.0264 -
val_acc: 0.9912
Epoch 8/15
259200/259200 [==============================] - 15s 60us/step - loss: 0.0274 - acc: 0.9908 - val_loss: 0.0276 -
val_acc: 0.9903
Epoch 9/15
259200/259200 [==============================] - 15s 60us/step - loss: 0.0253 - acc: 0.9914 - val_loss: 0.0219 -
val_acc: 0.9925
Epoch 10/15
259200/259200 [==============================] - 16s 60us/step - loss: 0.0247 - acc: 0.9916 - val_loss: 0.0202 -
val_acc: 0.9931
Epoch 11/15
259200/259200 [==============================] - 16s 60us/step - loss: 0.0238 - acc: 0.9920 - val_loss: 0.0220 -
val_acc: 0.9922
Epoch 12/15
259200/259200 [==============================] - 16s 60us/step - loss: 0.0235 - acc: 0.9923 - val_loss: 0.0406 -
val_acc: 0.9867
Epoch 13/15
259200/259200 [==============================] - 15s 60us/step - loss: 0.0223 - acc: 0.9924 - val_loss: 0.0239 -
val_acc: 0.9923
Epoch 14/15
259200/259200 [==============================] - 16s 60us/step - loss: 0.0217 - acc: 0.9927 - val_loss: 0.0180 -
val_acc: 0.9941
Epoch 15/15
259200/259200 [==============================] - 16s 60us/step - loss: 0.0213 - acc: 0.9929 - val_loss: 0.0225 -
val_acc: 0.9927
```

3.1 Graphiques de la perte et de la précision

In [20]:
```
plt.clf() # clear figures

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





3.2 Entrainement avec toutes les données

In [21]:
```
model.fit(train_x,
          train_y,
          epochs=15,
          batch_size=128)
```

```
Epoch 1/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0207 - acc: 0.9930
Epoch 2/15
324000/324000 [==============================] - 17s 54us/step - loss: 0.0206 - acc: 0.9933
Epoch 3/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0196 - acc: 0.9936
Epoch 4/15
324000/324000 [==============================] - 17s 54us/step - loss: 0.0191 - acc: 0.9936
Epoch 5/15
324000/324000 [==============================] - 17s 54us/step - loss: 0.0183 - acc: 0.9939
Epoch 6/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0173 - acc: 0.9942
Epoch 7/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0176 - acc: 0.9941
Epoch 8/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0174 - acc: 0.9941
Epoch 9/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0168 - acc: 0.9943
Epoch 10/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0158 - acc: 0.9946
Epoch 11/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0166 - acc: 0.9944
Epoch 12/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0155 - acc: 0.9947
Epoch 13/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0144 - acc: 0.9951
Epoch 14/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0147 - acc: 0.9950
Epoch 15/15
324000/324000 [==============================] - 18s 54us/step - loss: 0.0150 - acc: 0.9949
```

Out[21]:
```
<keras.callbacks.History at 0x7f5f646f0250>
```

3.3 Sauvegarde du modèle

In [22]:
```
# Sauvegarde de l'entrainement
# architecture + poids + optimizer state

model.save('../INPUT_DATA/FilesForDataBase/deepsat6/sat6_4bands_trained_model.h5')
```

4. Evaluation du modèle

In [23]:
```
#Chargement du modèle
from keras.models import load_model

model = load_model('../INPUT_DATA/FilesForDataBase/deepsat6/sat6_4bands_trained_model.h5')


# Evaluation du modèle
results = model.evaluate(test_x, test_y)

print results
```

```
81000/81000 [==============================] - 3s 41us/step
[0.011760508163812981, 0.9960246913580247]
```

Annexe 2 - Dictionnaire des données du modèle physique

Dictionnaire des données du modèle physique			
data : Contient les informations sur les données disponibles.			
Nom du champ	**Type de données**	**Contraintes**	**Description**
data_id	Integer	Clé primaire	Identifiant unique généré automatiquement.
type	Varying Character		Type de jeu de données, prend soit la valeur "train" pour une donnée utilisée pour entrainer le modèle, soit la valeur "test" pour une donnée utilisée pour tester le modèle.
fname	Varying Character		Nom du fichier stocké sur le serveur qui contient les données (images). Le fichier doit être dans le format d'un tenseur.
fname_label	Varying Character		Nom du fichier contenant les étiquettes des données.
nb_image	Integer		Nombre d'images dans le jeu de données
dataset_id	Integer	Clé étrangère	Clé étrangère provenant de la table dataset_information. Pour chaque jeu de données, cette clé doit apparaître deux fois, une fois pour référencer le jeu de données d'entrainement et une fois pour le jeu de données de test.
dataset : Contient les informations sur les jeux de données.			
Nom du champ	**Type de données**	**Contraintes**	**Description**
dataset_id	Integer	Clé primaire	Identifiant unique généré automatiquement.
dataset_name	Varying Character		Nom du jeu de données.
nb_class	Integer		Nombre de classes dans le jeu de données
class_name	Text Array		Nom des classes du jeu de données.
folder_name	Varying Character		Nom du dossier sur le serveur qui contient les jeux de données.
tile_size	Integer		Taille de tuiles
spatial_res_m	Double		Résolution spatiale du jeu de données en mètres.
aquisition_date	Date		Date d'acquisition du jeu de données.
nb_band	Integer		Nombre de bandes des images du jeu de données.
bit_depth	Integer		Valeur maximale de la profondeur des couleurs. Par exemple, pour 8 bits, inscrire 255.
reference	Varying Character		Source de provenance des données.
url	Varying Character		Lien URL vers la provenance de données.

model : Contient les informations sur les modèles d'apprentissage profond disponibles dans la base de données.			
Nom du champ	**Type de données**	**Contraintes**	**Description**
model_id	Integer	Clé primaire	Identifiant unique généré automatiquement.
model_name	Varying Character		Nom du modèle.
fname	Varying Character		Nom du fichier stocké sur le serveur qui contient le modèle entrainé.
model_summary	Varying Character		Résumé de l'architecture du modèle. Contient les couches utilisées ainsi que le nombre de paramètres de chaque couche.
accuracy	Double		Précision du modèle avec les données de test.
loss	Double		Valeur de perte du modèle
nb_band	Integer		Nombre de bandes utilisées dans le modèle. Il est possible que le nombre de bandes soit inférieur au nombre de bandes disponibles dans la base de données. Par exemple, il est possible d'entrainer un modèle avec seulement les bandes RGB alors que les images du jeu de données possèdent les bandes RGBPIR.
band_name	Varying Character		Nom des bandes utilisées. Par exemple, RGB ou RGBPIR.
nb_train_img	Integer		Nombre d'images utilisées pour entrainer le modèle.
nb_test_img	Integer		Nombre d'images utilisées pour tester le modèle.
dataset_id	Integer	Clé étrangère	Clé étrangère provenant de la table dataset qui permet de lier un modèle avec les informations du jeu de données.
mosaic : Contient les informations sur les images enregistrées dans la base de données.			
Nom du champ	**Type de données**	**Contraintes**	**Description**
mosaic_id	Integer	Clé primaire	Identifiant unique généré automatiquement.
mosaic_name	Varying Character		Nom de l'image.
mosaic_path	Varying Character		Chemin d'accès sur le serveur vers le fichier de l'image.
spatial_res_m	Double		Résolution spatiale de l'image en mètres.
aquisition_date	Date		Date d'acquisition de l'image
nb_band	Integer		Nombre de bandes de l'image.
bit_depth	Integer		Valeur maximale de la profondeur des couleurs. Par exemple, pour 8 bits, inscrire 255.
reference	Varying Character		Source de provenance de l'image
url	Varying Character		Lien URL vers la provenance de l'image
geom	Geometry		Géométrie du contour de l'image.

prediction : Contient les prédictions des classes de l'image obtenues avec les modèles.			
Nom du champ	**Type de données**	**Contraintes**	**Description**
mosaic_id	Integer	Clé primaire et étrangère	Clé primaire avec mosaic _id, qui fait référence à la table mosaic.
model_id	Integer	Clé primaire et étrangère	Clé primaire avec model_id, qui fait référence à la table model.
tile_number	Integer	Clé primaire	Identifiant unique de la tuile par prédiction. Prend des valeurs de 0 à (nombre de tuiles - 1).
class	Double Array		Tableau de taille variable qui contient les valeurs des prédictions. Contient une entrée par nombre de classes dans le modèle dont la somme fait 1.
most_likely_class	Varying Character		Classe la plus probable.
geom	Geometry		Géométrie de la tuile.